VŠB – TECHNICKÁ UNIVERZITA OSTRAVA
Fakulta elektrotechniky a informatiky

DOKTORSKÁ DISERTAČNÍ PRÁCE

2011 **Mgr. Jan Pokorný**

VŠB – TECHNICKÁ UNIVERZITA OSTRAVA
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Metody ověřování mechanismu vstupního řízení real-time databázového systému s použitím jeho experimentálního modelu

Doktorská disertační práce

Zpracoval: **Mgr. Jan Pokorný**

Školitel: **Doc. RNDr. Jindřich Černohorský, CSc.**

Prosinec 2011

„Prohlašuji, že jsem disertační práci vypracoval samostatně s využitím uvedených pramenů a literatury“.

……..

Rád bych touto cestou poděkoval svému školiteli, Doc. RNDr. Jindřichu Černohorskému, Csc. a také Ing. Václavu Královi, Ph.D. za odbornou a osobní pomoc při zpracování této práce. Rovněž děkuji své rodině za trpělivost a podporu při realizaci práce a během samotného studia.

ANOTACE

Hlavním problémem souvisejícím s plánováním a zpracováním transakcí v databázových systémech reálného času je skutečnost, že potřebné zdroje systému pro zpracování jednotlivých transakcí nejsou před samotným zpracováním transakce známy. Základní motivací této práce je mezera ve výzkumech zabývajících se vhodným návrhem databází reálného času, konkrétně návrhem mechanismů vstupního řízení databázového systému zabraňujícího přetížení systému pro různé typy vstupních transakcí, přičemž by byl tento přístup založený na aktuálním stavu systému. Hlavním smyslem mechanismu vstupního řízení je sledovat vytížení systému a předejít saturaci, případně provést nezbytné kroky, pokud je systém přetížený. Je důležité zdůraznit, že protokoly vstupního řízení používané v databázových systémech reálného času vyžadují specifický přístup. To znamená, že známé mechanismy ze systémů reálného času nelze jednoduše aplikovat do prostředí databázových systémů reálného času. V této disertační práci byl představen a popsán vylepšený systém reálného času, Testbed, sloužící především pro testovací účely. Systém byl vyvinut na platformu reálného času, RTX. Navržený mechanismus vstupního řízení byl testován a porovnán s existujícími přístupy. Můžeme konstatovat, že výsledky výzkumu byly ověřeny na existující platformě databázového systému v reálném čase.

Klíčové slova

databázové systémy reálného času, experimentální databázový systém reálného času, Testbed, vstupní řízení, algoritmy vstupního řízení, ovladač zahlcení

ANNOTATION

The main issue related to transaction planning and execution in real-time database system is based on the fact that we are not able to determine system resources for transactions execution in advance. A primary motivation of this work is a gap in real-time database research, especially in area of admission control mechanisms used feedback control strategy. The main purpose of an admission control mechanism is to control a system utilization and to avoid system saturation or provide correction steps if the system is overloaded. It is necessary to highlight that admission control protocols used in real-time database system require really specific approach. It means that well-known mechanisms from classic real-time system are not fully applicable in the real-time database area. In this thesis has been introduced and described an improved experimental real-time database system, called Testbed. The system has been developed on RTX real-time platform and it is focused on testing purposes mainly. Suggested mechanism of admission control has been tested and compared with well known approaches. We can proclaim that research conclusions were verified on an existing real-time database platform.

Keywords

real-time database management system, experimental real-time database management system, Testbed, admission control, admission control strategy, congestion control

SEZNAM ZKRATEK

Akronym	Význam (ekvivalent či popis v českém jazyce)
2PL	Two-Phase Locking (dvoufázové zamykání)
2PL-STRICT	Two-Phase Locking Strict
2PL-WP	Two-Phase Locking Wait-Promote
AAP	Adaptive Access Parameter
AC	Admission Control (vstupní řízení)
ACID	Atomicity, Consistency, Isolation, Durability
AED	Adaptive Earliest Deadline
AEVD	Adaptive Earliest Virtual Deadline
API	Application Programming Interface
BM	Buffer Manager, modul systému Testbed
CC	Concurrency Control (souběžné řízení)
CCCC	C and C++ Code Counter (softwarový nástroj)
CCM	Concurrency Control Manager
CDF	Criticality Deadline First
CM	Catalog Manager, modul systému Testbed
CPU	Central Processing Unit
DBMS	DataBase Management System
DFD	Data Flow Diagram
DLL	Dynamically Linked Library (dynamicky linkovaná knihovna)
DM	Deadline Monotonic
EDF	Execute Deadline First
FIFO	First In, First Out
FRT	Firm Real-Time Transaction
GUI	Graphic User Interface
HAL	Hardware Abstraction Layer
HRT	Hard Real-Time Transaction
INI	Initialization file (inicializační soubor)
IM	Index Manager
I/O	Input/Output
IPC	Inter-Process Communication (meziprocesorová komunikace)
IRQ	Interrupt Request (požadavek přerušení)
ISR	Interrupt Service Request

Akronym	Význam (ekvivalent či popis v českém jazyce)
MCF	Most Critical First
MDARTS	Multiprocessor Database Architecture for hard Real-Time Systems
MMX	MultiMedia eXtensions
MP	Multiprocessor
MRTDBS	Mixed Real-Time Database System
MS	Microsoft
MSI	Message Signal Interrupt
OS	Operating System (operační systém)
POSIX	Portable Operating System Interface
RM	Rate Monotonic
RMS	Rate Monotonic Scheduling
RT	Real-Time (reálný čas)
RTDBS	Real-Time DataBase System (databázový systém reálného času)
RTDBMS	Real-Time DataBase Management System
RTOS	Real-Time Operating System (operační systém reálného času)
RTSS	Real-Time SubSystem, část RTX platformy
RTX	Real-Tme eXtension, produkt firmy IntervalZero
SAD	Systém Analýzy Dat, softwarový nástroj
SP	Service Pack (opravný balíček)
SRI	Service Request Interrupt
SRT	Soft Real-Time Transaction
SŘBD	Systém Řízení Báze Dat
SSD	Solid-State Drive
SSE	Streaming SIMD Extensions
Testbed	Experimentální databázový systém reálného času
TM	Transaction Manager
TRX	Transaction
USB	Universal Serial Bus
V4DB	Kódové jméno experimentálního systému Testbed verze 1.0
VxSim	VxWorks Target Simulator
WCET	Worst Case Execution Time
WinDbg	Microsoft Kernel Debugger

Obsah

ÚVOD

Během posledních let rapidně vzrůstají požadavky na zpracování stále většího množství dat do digitalizované podoby. Tyto požadavky jsou kladeny na široké spektrum aplikací. Tato disertační práci se věnuje problematice zpracování dat v reálném čase.

Oblast výzkumu databázových systémů reálného času je motivována širokým spektrem aplikačních oblastí. Jsou to například telekomunikační systémy, skladové systémy, systémy pro uchovávání a zpracovávání dat z finanční oblasti, specializované herní systémy či grafické systémy, které mohou využívat databázový systém reálného času k ukládání souřadnic bodů identifikující základní objekty používané k vykreslování a mnohé další aplikace řídících systémů. Obecně můžeme konstatovat, že nemusí jít pouze o využití databázových systémů reálného času provozovaných na skutečných platformách reálného času, ale že lze principů databázových systémů reálného času využít také v prostředí klasických informačních systému, s předem definovanými časovými odezvami na zpracování transakcí.

Aktuálnost tématu potvrzuje neustálý zájem o informace ohledně návrhu RTDBS a popis jednotlivých komponent od akademických pracovníků a vědců z celého světa. Například naposledy byl autor kontaktován vědcem Rajnish Kumar Choubey z Research Center Imarat v Indii.

Hlavním problémem souvisejícím s plánováním a zpracováním transakcí v databázových systémech reálného času je fakt, že potřebné zdroje systému pro zpracování jednotlivých transakcí nejsou před samotným zpracováním transakce známy. Tato skutečnost v kombinaci s možností zpracovávat různé druhy vstupních transakcí výrazně komplikuje možnost najít optimální řešení vstupního řízení, které by bylo maximálně efektivní.

Základní motivací této práce je mezera ve výzkumech zabývajících se vhodným návrhem databází reálného času, konkrétně efektivního návrhu mechanismů vstupního řízení databázového systému zabraňujícího přetížení systému pro různé typy vstupních transakcí, přičemž by tento přístup byl založený na aktuálním stavu systému. Měli bychom zdůraznit významný fakt, že vstupní řízení v databázových systémech reálného času vyžaduje zvláštní přístup, přičemž nelze vždy jednoduše aplikovat klasické přístupy, které známe z aplikací reálného času.

Cílem této disertační práce je navrhnout vstupní řízení, které dokáže efektivně předcházet přetížení systému. Úkolem je otestovat tento přístup a ukázat jeho výhody či nevýhody a dále pomocí testů vyhodnotit a porovnat se systémem bez vstupního řízení.

Hlavním přínosem této práce je návrh a otestování vstupního řízení, které je založeno na zpětné vazbě systému, tzn. podle aktuálního stavu systému má schopnost reagovat, zabránit přetížení systému a přitom maximalizovat počet zpracovávaných transakcí v systému. Následně umožní regulovat počet transakcí na vstupu systému v případě, že frekvence příchozích transakcí přesáhne kapacitu systému. Předcházet přetížení systému i případným nezpracováváním vstupních transakcí je nutné, pokud chceme zabránit zahlcení systému a degradaci výkonnosti celého systému. V disertační práci je dále navržen a popsán přístup pro monitorování aktuálního stavu systému.

Výsledky práce umožní efektivněji využít možnosti databázového systému reálného času, přičemž systém se správně zvoleným algoritmem vstupního řízení dokáže lépe reagovat na vstupní zátěž a efektivně předcházet saturaci systému.

1. CÍLE A PŘÍNOS PRÁCE

V následujících kapitolách jsou popsány cíle práce, oblast řešení a přínos práce.

1.1 CÍLE PRÁCE

Cílem je teoreticky navrhnout a na základě experimentů potvrdit či vyvrátit, zda vstupní řízení založené na aktuálních informacích ze systému je efektivní, případně za jakých okolností bude navrhovaný přístup podávat lepší výsledky. Cíle disertační práce lze strukturovat do následujících oblastí:

(1) Analýza a návrh vstupního řízení založeného na zpětné vazbě systému

- Analýza problematiky vstupního řízení a existujících algoritmů dle typů vstupních transakcí
- Návrh vstupního řízení
- Příprava testovacích případů a definice jednotlivých metrik

(2) Vývoj testovacího prostředí

- Vývoj experimentálního databázového systému reálného času pro možnost ověření navržené metody vstupního řízení
- Implementace a integrace vstupního řízení do databázového systému reálného času Testbed

(3) Ověření

- Zátěžové testování a zjištění stavu systému, kde je již znatelný vliv degradace na propustnost
- Specifikace testovacích případů a vstupních parametrů a metrik dle výsledků základních testů
- Testování algoritmu vstupního řízení pro variabilní parametry systému
- Zjištění vlivu monitorovacího modulu a vstupního řízení na chod systému

1.2 OBLASTI ŘEŠENÍ

Analýza vstupního řízení pro oblast RTDB

Analytická část práce zahrnuje teoretický rozbor, za jakých podmínek a kdy dochází k přetížení systému pro různé vstupy do databázových systémů reálného času. Dále teoretický rozbor popisuje za jakých okolností je vhodné použít existující algoritmy vstupního řízení a přidělování priorit.

Vstupní řízení transakcí

Navrhnout takový mechanismus vstupního řízení, který bude schopen předcházet přetížení systému, přičemž bude tento systém založen na zpětné vazbě, tzn. podle aktuálního stavu systému.

Ověření funkce navrženého vstupního řízení

Na základě definic testovacích scénářů připravit na experimentálním systému databázového reálného času simulace přetížení systému Testbed. Dle vykonaných testů a výsledků navrženého vstupního řízení ukázat, kdy je vhodné používat jednotlivé druhy vstupního řízení a za jakých vstupních podmínek. Popsat vhodné postupy při nastavení jednotlivých komponent databázového systému reálného času.

1.3 PŘÍNOS PRÁCE

Přínos práce lze shrnout do následujících bodů:

(1) Analýza a návrh algoritmu vstupního řízení založeného na zpětné vazbě systému. Vyhodnocení aplikace vstupního řízení na základě provedných testů. Zjištění vlivu vstupního řízení na činnost databázového systému reálného času systému dle vykonaných experimentů.

(2) Vytvoření testovací platformy, kterou lze jednoduše použít k testování existujících či nově navržených algoritmů v jednotlivých komponentách a možností systém dále rozšiřovat. Systém poskytuje možnost efektivněji získávat informace a měřit chování systému během zpracování vstupních transakcí. Systém Testbed lze považovat za základ pro další výzkum a možnost rozšíření, a to v mnoha oblastech, které dále popisuje kapitola č. 8..

1.4 STRUKTURA PRÁCE

Práce je rozdělena do následujících částí, které na sebe logicky navazují. Následující text představuje stručný přehled jednotlivých kapitol.

Teoretický rozbor – přibližuje základní pojmy RTDBS, mimo jiné transakce jako vstupy do systému a jejich atributy v prostředí reálného času. Dále popisuje výzkumy v oblasti databázových systému reálného času.

Způsob řešení – popisuje jak autor při tvorbě disertační práce postupoval.

Postup řešení – ukazuje kroky a aspekty, kterými autor během zpracování práce procházel.

Testování vstupního řízení a validace systému Testbed – obsahuje popis provedených testů a následných komentářů. Informace, jaké byly zvoleny metriky a nastavení testovacího prostředí Testbed. Popis jak byl systém před jednotlivými testy verifikován a validován.

Výsledky a vyhodnocení – vyhodnocení vstupního řízení, výsledků jednotlivých testů a práce jako celku.

Celkové shrnutí – obsahuje závěrečné shrnutí celé práce a jejích přínosů pro navazující výzkum.

Další možné rozšíření a využití systému – přibližuje, jak lze využít zjištěných informací, jak vyvinutý systém nadále rozvíjet či využít k dalším testům nebo specifickým aplikacím.

2. TEORETICKÝ ROZBOR

2.1 VÝZKUM DATABÁZOVÝCH SYSTÉMŮ REÁLNÉHO ČASU

Oboru databázových systémů reálného času je věnována pozornost od počátku devadesátých let minulého století (1). Od té doby probíhá výzkum, který se snaží o integraci specifik systémů reálného času na jedné straně a databázových systémů na straně druhé. Hlavním zaměřením výzkumu v oblasti databázových systémů reálného času jsou vnitřní algoritmy nejdůležitějších částí databázových systémů, ať už se jedná o mechanismy přidělování priorit (2), (3), souběžného řízení (1), (4), indexování dat (5) nebo přístupy pro ukládání dat do paměti (6). Mnohdy jsou přístupy z klasických databázových systému adaptovány pro použití v prostředí reálného času (7), (8), existují však také zcela nové algoritmy a přístupy pro mechanismus vstupního řízení (9), (10).

K výzkumu databázových systémů reálného času lze přistupovat různými způsoby. Krátce přibližme projekt RTDBS pro hard real-time systémy s názvem MDARTS (11), kdy je celý přístup velmi zjednodušen pomocí zajištění specifických vstupních transakcí. Transakce obsahuje pouze jednu databázovou operaci. Systém nepředpokládá, že dojde ke konfliktům při přístupu k datům, tzn. nejsou využity žádné způsoby zamykání spojené se souběžným řízením transakcí. Kritická sekce je v multiprocesorovém systému hlídaná kruhovou zamykací frontou a navíc je zamykání řízeno systémem, ne samotnou aplikací databázového systému. Pomocí takového systému může RTDBS garantovat dobu provádění transakcí. Systém je schopen provést pohotovostní akce, pokud nejsou transakce zpracovány v rámci kritického termínu.

Přestože je oblast real-time databázových systému velmi specifická, existují základní poznatky z výzkumů vstupního řízení transakcí pro oblast reálného času. Z obecného pohledu jsou velmi přínosné práce (12), (1), (13) a dále vědecké práce, které vznikly na Univerzitě ve Virginii v rámci projektu „Real-Time Database Systems Research“. Základní typy zkoumaných mechanismů vstupního řízení jsou popsány v kapitole č. 2.4. Na druhé straně byl prezentován framework[1] zajišťující vstupní řízení v úzké spolupráci s mechanismy souběžného řízení (14).

Za další zajímavý a inspirativní projekt lze považovat projekt mixed RTDBS (7), který zajišťuje souběžnost transakcí a maximální blokovací dobu transakcí. U tohoto systému stojí za pozornost použité technologie, využití programovacího jazyka C++ a ryze objektového přístupu. Tato hard RTDBS je úzce spjata s operačním systémem reálného času a s procesy operačního systému, přičemž umožňuje výlučný přístup k datům a nepoužívá klasický relační databázový model.

Je zřejmé, že při návrhu RTDB systémů je nutné zahrnout charakteristiky transakcí reálného času do rozhodovacího mechanismu každé z komponent. Jinými slovy, při výzkumu vlivu algoritmů jednotlivých komponent je nutné brát v potaz i všechny další spolupracující části, moduly. Stejně důležitou součástí je výzkum propojení modulů databázového zpracování s prostředky operačního systému reálného času. Například výzkum optimálního plánování databázových transakcí (15), podobně jako plánování úloh operačního systému reálného času nebo zakomponování procesních priorit do algoritmů zpracování transakcí. V případě řešení konfliktů v RTDBS se jedná výhradně o adaptaci mechanismů klasických databázových systémů. Jistou výjimkou je specifický přístup pro distribuované databázové systémy reálného času (16). Z nalezené literatury, na kterou se tato práce

[1] Pojem *framework* má český ekvivalent *rámec*, o kterém autor textu ví. Přesto si myslí, že v pojem *framework* je obecně známější a byl přijat odbornou veřejností, a proto není tento pojem v textu přeložen.

odkazuje, lze zjistit mnoho zajímavých závěrů jednotlivých výzkumů, které jsou hodnotnou inspirací pro budoucí výzkum.

Výzkum probíhá z větší části v teoretické rovině s využitím simulací založených na matematických modelech. Tyto simulace mnohdy testují samostatné části databázového systému reálného času. V těchto případech se však přehlíží nutnost sledovat spolupráci jednotlivých modulů, komponent systému. Provádět reálné testy a testovací scénáře na skutečné platformě reálného času se snaží pouze velmi málo výzkumných týmů, viz výsledky výzkumu v (17).

Při výzkumu a vývoji databázových systémů reálného času nelze přehlédnou technologické hledisko. Existují doporučení pro technologie, které by se neměly využívat při návrhu soft real-time databázového systému. Například pro komunikaci mezi klienty a samotným systémem nepoužívat technologii remote procedure call, protože technologie prodlužuje dobu zpracování transakcí o 1-20 ms (18), i když v dnešní době můžeme využívat alternativní technologii binární RPC[2]. Je tedy nezbytné při návrhu systému také zvažovat technologické a aplikační hledisko.

2.2 DATABÁZOVÉ SYSTÉMY REÁLNÉHO ČASU

Operační systém reálného času (anglicky **Real-Time Operating System,** zkratka RTOS) je takový systém, který poskytuje nástroje, respektive prostředí pro provozování aplikací reálného času. **Aplikace reálného času** jsou pak takové aplikace, které kladou nároky nejen na funkční správnost, ale splňují i požadavky na provedení funkčních požadavků v čase. Požadavek na ohodnocení času reakce záleží na typu systému reálného času, viz kapitola č. 2.2.2. Důležitým hlediskem úspěšnosti nasazení systému není průměrný čas reakcí, ale čas reakce v tom nejhorším případě (anglicky **Worst Case Execution Time**, zkratka **WCET**) (19).

Databázový systém reálného času (anglicky **Real-Time Database System**, zkratka **RTDBS**) nazýváme systém, který zpracovává **transakce**[3] s přesně definovanými požadavky na samotné zpracování. Databázový systém reálného času je tedy systém, jenž realizuje vstupní transakce s explicitně definovanými nároky na dobu jejich provedení, přičemž pracuje s daty, která mají definovaný časový interval, ve kterém jsou považována za platná (20).

Obecně můžeme specifikovat dva základní rozdíly mezi klasickými databázovými systémy a databázovými systémy v reálném čase (18):

1. Transakce v prostředí reálného času obsahují časové podmínky, podle typu aplikace. Tyto časové podmínky mohou být **hard** nebo **soft** a nebo hodnota dat lineárně klesá dle odezvy po překročení kritického termínu viz kapitola č. 2.2.1. Toto rozdělení je analogické s **hard**, **soft** či **firm** úlohami v prostředí reálného času.
2. Data, získaná v odpovědi na databázovou operaci mají **absolutní** a **relativní konzistenci**, více viz následující kapitola.

[2] ERIC KIDD. *XML-RPC for C and C++* [software]. [přístup 15. prosince 2011]. Dostupné z http://xmlrpc-c.sourceforge.net/.

[3] Více v kapitole 2.2.1.

2.2.1 Transakce v prostředí reálného času

Základní jednotkou zpracování v RTDB systémech je stejně jako v případě klasických databázových systémů řízení báze dat tzv. **Transakce** (zkratka **trx**) (21), (22), neboli množina databázových operací. Pokud hovoříme o transakčním zpracováním dat, musí být zajištěny tzv. **ACID vlastnosti**:

- **A** – **Atomicita** (anglicky **Atomicity**) zaručuje, že se transakce provede celá nebo se neprovede vůbec. Atomicitu transakce lze zajistit pomocí následujících metod: metody zpožděné aktualizace, metody přímé aktualizace nebo metody stínového stránkování (21).
- **C** – **Konzistence** neboli **soudržnost** (anglicky **Consistency**). Pokud je transakce ukončena, musí dojít k navrácení databáze do konzistentního stavu. Specifikujeme interval soudržnosti datových proměnných a rozlišujeme **absolutní a relativní soudržnost** (18).
- **I** – **Izolovanost** (anglicky **Isolation**). Operace uvnitř transakce jsou skryty před vnějšími operacemi.
- **D** – **Trvalost** (anglicky **Durability**). Pokud je transakce úspěšně ukončena, data v databázovém systému jsou trvale pozměněna.

Pro zajištění výše uvedených vlastností jsou při transakčním zpracování používány operace:

- **BEGIN** – specifikuje začátek transakce.
- **COMMIT** – zajistí ukončení transakce, uložení dosažených výsledků během zpracování jednotlivých databázových operací do databáze.
- **ROLLBACK** – odvolání změn, není-li definován tzv. **Save Point** (místo, po kterém lze provedené změny vrátit zpět), tak vrací databázi do nějakého předchozího stavu před započetím vykonávání transakce. Operace **Rollback** je velmi důležitá pro zachování datové integrity.

RTDBS jsou velmi specifickým druhem databázových systémů, u kterých se můžeme v závislosti na konkrétní aplikaci vyhnout zajišťování požadavků na ACID vlastnosti, což může v určitých situacích snížit režii systému. Můžeme například omezit trvanlivost dat, pokud takové informace pro nás ztrácejí v čase význam nebo nemusíme vyžadovat sériovou soudržnost (21).

Transakce přistupuje k datům, která mají hodnoty a platnost měnící se v čase. Správnost dat a jejich integrita je zajištěna díky metodám, efektivním přístupům k datům a dalším technikám, které zajišťují garanci korektního zpracování transakcí při souběžném zpracování a korekci chyb při zpracování (12).

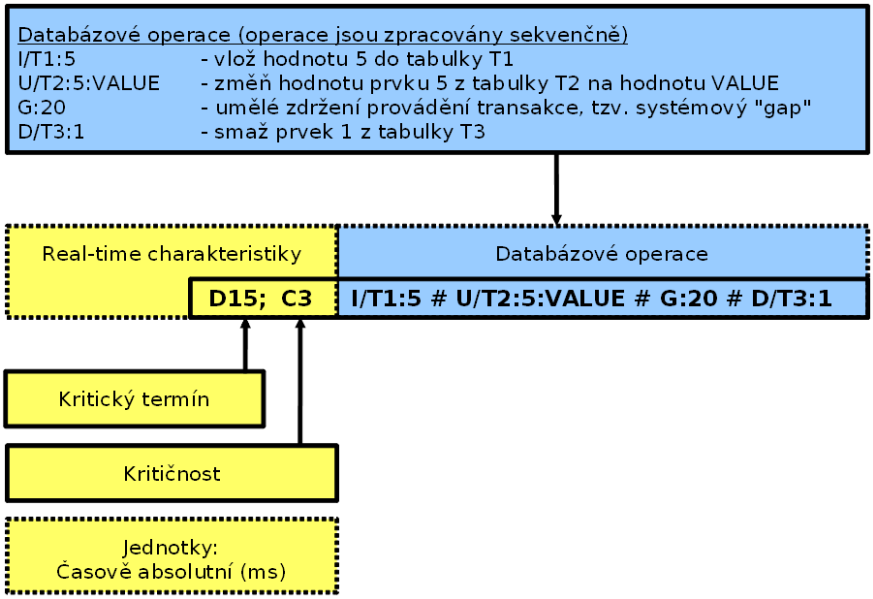
Pojem transakce v kontextu reálného času má stejný význam jako v oblasti klasických databázových systému. Existuje zde ale jeden významný rozdíl. **Transakce** zpracovávané v **prostředí reálného času** či **transakce reálného času** (ekvivalent **real-time transakce**) jsou rozšířeny o tzv. **charakteristiky reálného času** (anglicky **real-time characteristics**), neboli **časová omezení**, která mohou ovlivnit zpracování dané transakce.

Typické charakteristiky reálného času, které známe při vstupu transakce do systému:

- **kritický termín** (anglicky **deadline**) – čím nižší je kritický termín, tím vyšší je požadavek na rychlejší zpracování transakce
- **kritičnost** (anglicky **criticalness**) – parametr popisuje důsledky promeškání termínu transakce

Počet vstupních charakteristik úzce souvisí s vnitřními algoritmy databázového systému reálného času. S transakcí mohou být spojeny charakteristiky, ze kterých se při vstupu do systému vypočítá priorita úlohy pro zpracování dané transakce, případně se používá kombinace více charakteristik, jak je popsáno v kapitole 2.4.1.

Příklad **transakce reálného času** vstupující do databázového systému reálného času[4]:



Obrázek 2.1: Popis struktury transakce reálného času

Časová omezení transakcí jsou **relativní**, přičemž nelze předem zjistit dobu vykonání libovolného typu transakce, protože nejsme schopni zjistit zdroje systému nutné pro zpracování.

Ve vyvíjeném systému Testbed viz kapitola 3.1, lze časové jednotky definovat podle nastavení systému v milisekundách nebo systémových časových jednotkách (anglicky time ticks). V systému se berou v potaz dvě časová omezení, kritický termín a kritičnost.

[4] Obrázek byl přepracován a převzat z (20).

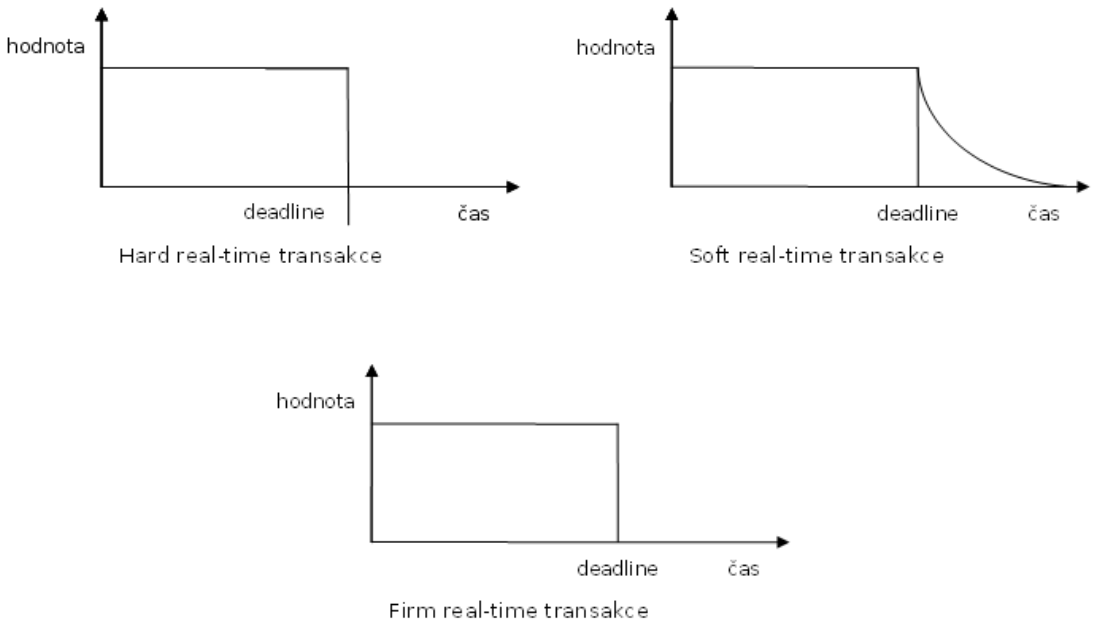
Za **aktivní transakci** považujeme takovou transakci, která je zpracovávaná databázovým systémem reálného času.

Transakce reálného času by měla být dokončena do svého kritického termínu. Obecně řečeno by měly být splněny všechny real-time charakteristiky, aby byl zajištěn maximální přínos transakce pro systém. Takový požadavek je však velmi obtížné splnit. V případě, že transakce svůj kritický termín nesplní, hovoříme o **zpožděné transakci**. Nesplnění kritického termínu nemusí nutně znamenat katastrofální důsledky na systém, protože závisí na typu transakce, která je dokončena po kritickém termínu.

Transakce lze členit podle nároků na splnění aplikovaných časových požadavků do tří skupin, obdobně jako tradiční systémy reálného času (23):

- **tvrdé** (anglicky **hard real-time transaction**, zkratka **HRT**) – promeškání kritického termínu hard transakce reálného času může vyústit v katastrofu, proto má tato transakce velmi negativní hodnotu pro systém po překročení kritického termínu.
- **přísné** (anglicky **firm real-time transaction**, zkratka **FRT**) – Firm transakce je speciálním případem soft transakce, její přínos pro systém je v okamžiku kritického termínu nulový.
- **měkké** (anglicky **soft real-time transaction**, zkratka **SRT**) – Soft transakce má určitou pozitivní hodnotu i po nesplnění časového omezení, tato hodnota však pomalu klesá k nule.

Matematicky můžeme znázornit typy transakcí jako funkce hodnoty v čase následovně:



Obrázek 2.2: Funkce hodnoty v čase pro různé kategorie transakcí

Dále můžeme typy transakcí členit podle typu databázových operací, viz (18):

- **External-input** – tento typ transakcí ukládá informace o externích událostech do databáze, tento typ transakcí bývá obvykle **zapisovací** (anglicky **write-only**)
- **External-output** – tyto periodické nebo aperiodické[5] transakce získávají informace ze systému a jsou většinou **čtecí** (anglicky **read-only**)
- **Internal** – mohou být vnořené, případně volat jinou transakci

Přerušení Transakce

Transakce je přerušená na základě akce od uživatele či systému samotného. Obecně rozlišujeme dva typy přerušení transakcí, **ukončující** a **neukončující** (18):

1. **Přerušení s ukončením**

   Transakce jsou přerušeny s ukončením, jestliže již nejsou dále zpracovávány systémem. K tomuto stavu dochází z následujících důvodů:

   - z důvodu promeškání či nesplnění kritického termínu či ostatních real-time charakteristik
   - z důvodu výjimečné události, přerušení provedené systémem

2. **Přerušení bez ukončení**

   Transakce je po přerušení znova spuštěna, přičemž je zajištěna konzistence dat. Tuto aktivitu nazýváme pojmem, že transakce byla **restartována**. Transakce obecně může být i několikrát přerušena a znovu začleněna do vstupní fronty transakcí, přičemž může mít pokaždé jinou prioritu na základě vnitřních algoritmů systému. Restartovaná transakce musí opět projít na vstupu modulem **vstupního řízení**, viz kapitola č. 2.4.

   K přerušení transakcí bez ukončení může docházet z následujících důvodů:

   - nastane z důvodu **uváznutí**, tzn. dojde ke sporu o zámek s jinou transakcí, jenž má vyšší prioritu (anglicky **deadlock**)
   - **konfliktu na datech**, vykonávání transakce může být i v tomto případě zdrženo a následně přerušeno jinou transakcí

O **přerušení** či případném **ukončení** zpracování transakce se rozhoduje v bloku vstupní řízení. Modul databázového systému s názvem vstupní řízení dle použitého algoritmu rozhodne, zda je systém dostatečně vytížený a zda lze zaručit úspěšné zpracování aktuálně vstupující transakce do systému. V souladu s typem transakce a algoritmu vstupního řízení rozhodne, zda-li bude daná transakce do systému vpuštěna ke zpracování, ukončena nebo restartována.

RTDBS jsou především vyhodnocovány podle toho, jak často transakce promeškávají svůj kritický termín, jaké je průměrné zpoždění transakcí neboli obecně, jaké jsou důsledky promeškání kritických termínů transakcí.

[5] Aperiodické transakce jsou takové transakce, které jsou generovány náhodně.

Odhad časové odezvy transakcí

V případě RTDBS však není možné vždy predikovat dobu nutnou pro zpracování transakce. Na rozdíl od klasických real-time aplikací zde existuje několik faktorů, které ztěžují odhad časové odezvy transakcí, více viz (20):

1. **Závislost na hodnotách dat**

   Provedení transakce může být závislé na datech získaných z databáze nebo na vypočítaných hodnotách z těchto dat. V systému se data dynamicky mění, proto nelze předem jednoznačně predikovat dobu provedení transakce.

2. **Konflikty na datech a zdrojích systému**

   Plně transakční přístup znamená využití mnoha databázových protokolů, jejichž časové odezvy jsou nepředvídatelné. Např. protokoly souběžného řízení často používají blokování jiných transakcí a jejich opětovné restartování, navíc často s kaskádovitým efektem, kdy restart jedné transakce může způsobit restartování ostatních blokovaných transakcí, což způsobuje další zpoždění. Rovněž požadavek zajištění ACID vlastností může mít za následek velkou režii systému, což odhad dále ztěžuje. Garance absolutního splnění termínu by znamenala enormní předimenzování zdrojů.

3. **Dynamické stránkování a I/O operace**

   Data jsou obvykle uchovávána na sekundárních zařízeních, např. na pevném disku. Požadavek na získání konkrétních dat znamená nutnost načtení několika stránek do paměti a po vykonání transakce jejich uložení zpět. Někdy však již jsou požadovaná data v paměti uložena a spravována pomocí modulu **buffer management**, což samozřejmě snižuje režii systému a urychlí provedení databázové operace. Okamžitou dostupnost dat ovšem nelze předpokládat.

4. **Závislost na velikosti databáze**

   Transakční přístup může být závislý na velikosti databáze, tzn. struktuře a počtu uložených dat. Jestliže máme transakci, která provádí operace na rozsáhlé databázi, lze předpokládat, že doba provedení transakce bude delší, než na databázi malé.

2.2.2 Typy databázových systémů reálného času

Analogicky s členěním transakcí, členíme také databázové systémy reálného času (24):

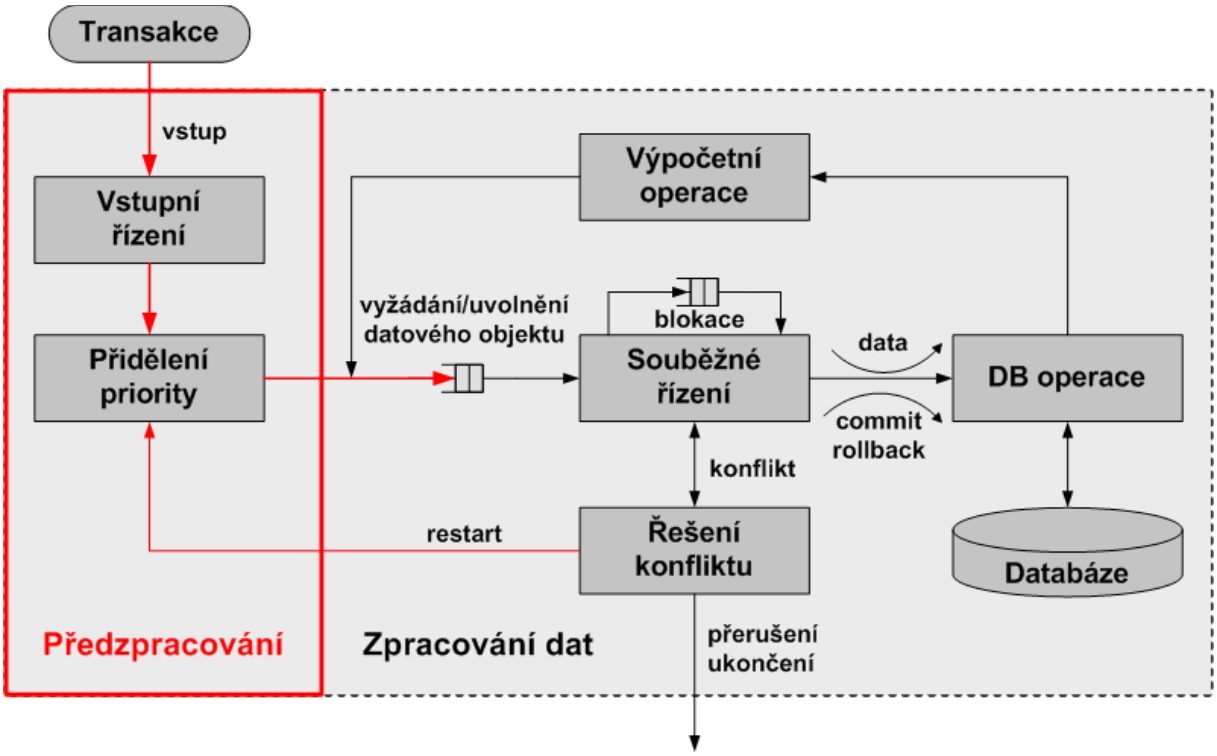
- Za **nepoddajný** (anglicky **hard**) databázový systém reálného času považujeme takový systém, jehož vstupem jsou pouze transakce typu hard**.** Pokud transakce vstoupí do tohoto RTDB systému a je postoupena dalšímu zpracování, musí být garantováno, že bude vykonána.
- Za **měkký** (anglicky **soft**) databázový systém reálného času označujeme systém, který přijímá pouze transakce typu **soft**, tzn. aperiodické transakce.
- **Smíšený**[6] databázový systém reálného času (anglicky **mixed**, zkratka **MRTDBS** (7)) nazýváme takový systém, který na vstupu přijímá různé druhy transakcí.

[6] Jde o překlad autora z anglického pojmu. Autor si není vědom, že by existoval ekvivalentní český pojem.

2.2.3 Obecný model RTDBS

Databázový systém reálného času se skládá z množiny vzájemně integrovaných procesních komponent. Mezi nejdůležitější moduly patří vstupní řízení a přidělování priorit, mechanismus souběžného řízení a řešení datových konfliktů, detektor uváznutí, mechanismy pro restart transakcí, metody obnovy dat a přístupu na disk. Transakce prochází těmito komponentami pomocí definovaných operací až do okamžiku svého ukončení, úspěšného ukončení pomocí akce **commit** nebo neúspěšného pomocí akce **abort**, **rollback** nebo **restart**.

Databázový systém reálného času přijímá požadavky klientů, které mají formu transakcí. Klienti mohou být operátoři, řídící či informační systémy, které pracují s daty databázového systému reálného času.



Obrázek 2.3: Obecný model real-time databázového systému

Model z obrázku č. 2.3 je převzat a přepracován z publikace (13). Schéma slouží k popisu základních modulů a také principů transakčního zpracování v rámci databázových systémů reálného času. Červeně zvýrazněná oblast databázového systému reálného času je část systému, která je zkoumána v této disertační práci. Jednotlivé komponenty systému jsou popsány v kapitole 3.1.

2.3 PŘETÍŽENÍ SYSTÉMU

K přetížení databázového systému reálného času dochází z důvodu velkého množství transakcí, které vstoupily či vstupují do systému, získávají data či vzájemně soupeří o data, což odpovídá současné vysoké zátěži všech systémových zdrojů, procesoru, bufferů paměti a I/O front. Taková situace může mít v případě transakcí typu **hard** systémů katastrofální následky. Proto i v případě přetížení musí být zajištěna přednost provedení transakcí, které jsou pro systém nejkritičtější.

Přetížení systému je způsobeno následujícími faktory (18):

- **Požadavkem na zajištění sériové soudržnosti**, neboli konfliktem na požadovaných oblastech dat. Transakce může být zpožděná z důvodu čekání na data, která jsou právě uzamčena jinou transakcí. Jako řešení lze použít podobný mechanismus, jaký využívají operační systémy reálného času, zajištěním tzv. **dědění priorit**[7].

- **Konfliktem fyzických zdrojů**, a to i v případě, že neexistuje žádný konflikt o data. Spor o fyzické zdroje nastává tehdy, pokud je mnoho transakcí svázáno v čekajících frontách. V tomto případě lze situaci opět řešit mechanismem dědění priorit jednotlivých transakcí na úrovni operačního systému reálného času.

- **Zvýšením režie nutné pro správu transakcí,** což je velice opomíjeno v teoretických přístupech. V praktických testech je nutné sledovat režii přepínání kontextu a správy jednotlivých vláken a procesů, která narůstá s přibývajícím počtem transakcí a komplexností samotného databázové systému.

- **Používáním pevného disku a paměti.** V tomto případě je nutné používat speciální algoritmy pro před-načítání (anglicky **data buffering**) dat do paměti a snažit se minimalizovat přístupy na pevný disk. Problém s relativně dlouhou přístupovou dobou k pevnému disku lze mimo jiné řešit využitím **Solid-State Drive** (zkratka **SSD**) disků[8].

2.3.1 Důsledek přetížení systému

Vlivem tzv. **trashingu** propustnost systému lineárně klesá. V takovém případě se výrazně snižuje výkonnost systému a tím se snižuje počet transakcí splňujících kritické termíny či obecně časová omezení.

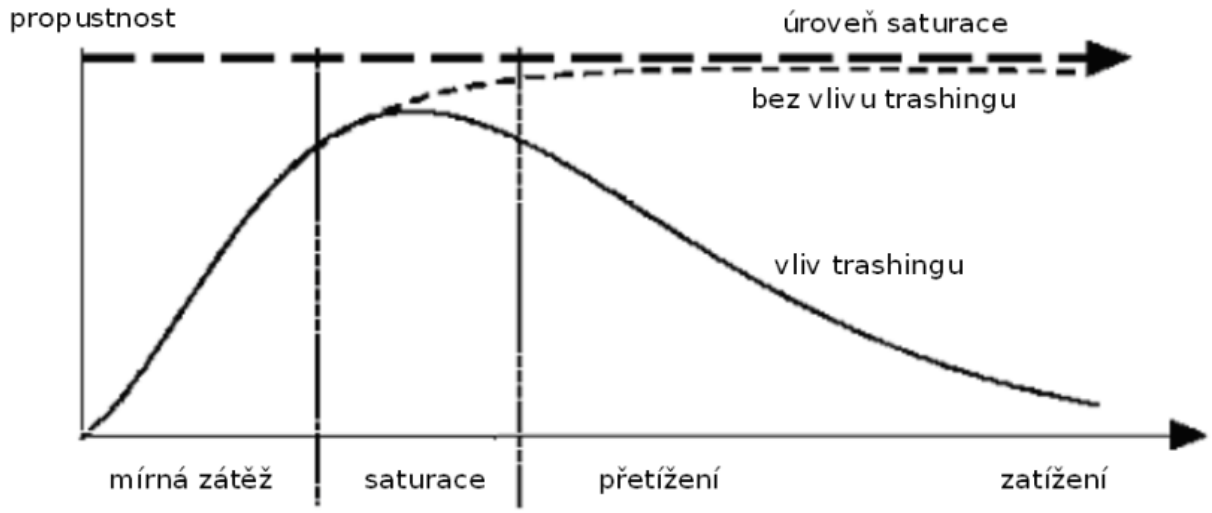
Důsledek tohoto fenoménu lze velmi dobře vidět na obrázku č. 2.4, který je znám jako **load-throughput function** (4). Vyjadřuje míru propustnosti v závislosti na zatížení obecného systému. Propustnost se při postupně narůstající zátěži zvyšuje téměř lineárně. Jakmile však je využita maximální kapacita systému, dochází k saturaci a další zvyšování zátěže vede k výraznému poklesu propustnosti, tzn. poklesu výkonnosti systému jako celku.

---

[7] YODAIKEN, Victor. *Against priority inheritance* [online]. 2002 [cit. 2011-12-15]. Dostupné z: http://www.linuxfordevices.com/files/misc/yodaiken-july02.pdf.

[8] AURORA, Valerie. *Log-structured file systems: There's one in every SSD* [online]. 2009 [cit. 2011-12-15]. Dostupné z: http://lwn.net/Articles/3511/.

„Trashing efekt“ byl popsán v (4) a základní přehled o různých přístupech pro oblast vstupního řízení například ve virtuálních úložných systémech lze nalézt v (25). Trashing v databázových systémech, spojený se zamykáním dat byl popsán v (26). Zjednodušený model byl také využit v (27) pro analyzování optimistického protokolu.



Obrázek 2.4: Propustnost databázového systému reálného času v závislosti na zatížení

Přetížením systému popisujeme stav, nemůže-li plánovač (anglicky **scheduler**) vyhovět všem kritickým termínům, říkáme, že je **systém přetížen**. Jinak řečeno za **přetížení databázového systému reálného času** považujeme takový stav systému, kdy je s přibývajícím počtem vstupních transakcí do systému degradována propustnost celého systému, tzn. přibývá procento neúspěšně zpracovaných transakcí.

Obecně rozlišujeme dvě oblasti související s přetížením systému - **oblast předcházení přetížení** a **zotavení z přetížení**.

2.3.2 Techniky pro předcházení přetížení

V této kapitole budou popsány techniky, jak předcházet přetížení systému, tzn. budeme se zabývat problémem saturace systému. Existují následující základní techniky pro regulaci zátěže *RTDBS* a předcházení přetížení systému:

- **‘Fixed upper bound’** definuje maximální počet souběžných transakcí, kdy je tento počet nastaven jako systémový parametr. Nejjednodušší přístup, který je vhodný pouze pro RTDB systémy s definovanými vstupními transakcemi. Určení vhodné hodnoty pro systémový parametr je podmíněno mnoha testy a lze jej použít v případě, že budeme znát dostatečně reprezentativní vzorek vstupních transakcí.

- Teoretické pravidlo nazývané **'rules of thumb'**. Analytické modely navrhují podmínky, které musí být dodrženy pro prevenci trashingu. V (26) lze nalézt tvrzení, že prevenci trashingu lze zajistit při platnosti vzorce:

$$\frac{k^2 n}{D} < 1.5$$

k – je počet datových prvků, ke kterým přistupuje daná transakce

n – je úroveň souběžnosti

D – velikost databáze

- Lze nalézt tvrzení, že střední hodnota konfliktů na jednu transakci nesmí přesáhnout koeficient 0,75. V takovém případě by mohla být prevence proti přetížení jednoduchá. Protože však nejsou detailní příklady použití tohoto a předchozího pravidla a nebyla provedena ověření, je nutno brát nasazení těchto pravidel s velkou obezřetností.

- Techniky založené **na úpravě algoritmu přidělování priorit**, nejčastěji úpravou algoritmu **EDF**. Nejznámější jsou algoritmy **AED**, **AEVD** nebo **AAP**. Některé z těchto technik byly popsány v kapitole 2.4.2.

- Úprava obecných přístupů založených **na zpětné vazbě systému** (anglicky **Feedback Control**) používaných v real-time aplikacích. U těchto typů vstupního řízení je dynamicky monitorováno chování systému. Řídicí mechanismus dynamicky reaguje na chování systému. Informace o těchto přístupech jsou známy pro obecné real-time systémy, ne však pro databázové systémy reálného času.

2.3.3 Techniky pro zotavení systému z přetížení

Jestliže dojde k přetížení systému, musí systém na tento stav reagovat. Oblast řešení se nazývá **overload management**, přičemž se rozhoduje, které transakce restartovat, tzn. měnit jejich priority pro zpracování či transakci ukončit. Obecně můžeme shrnout základní fakta o zpracování přetížení následovně (24):

Rozvrhovací algoritmy navržené pro hladkou degradaci systému za podmínek přetížení se snaží přiřadit nějakou míru důležitosti ke každému kritickému termínu transakce. Efektivní algoritmy umí řídit rozvrhování při přetížení tak, aby se minimalizovala škoda. Naneštěstí přitom rozvrhovací algoritmus ztratí příliš mnoho času a může počítat rozvrh delší dobu, než po jakou ho posléze provádí.

Různé typy vstupních transakcí mají různé požadavky na nastavení systému a zpracování. Lze konstatovat, že nelze navrhnout optimální algoritmus pro zotavení z přetížení systému s aperiodickými vstupními transakcemi (24). V následujících kapitolách budou teoreticky popsány oblasti zahrnující **předzpracování transakcí - vstupní řízení** a **přidělování priorit**.

2.4 VSTUPNÍ ŘÍZENÍ

Databázový systém reálného času přijímá požadavky ve formě transakcí reálného času a tyto požadavky zpracovává při vstupu do systému modul s názvem **vstupní řízení**. V následujících kapitolách se budeme tedy věnovat části **vstupní řízení** (anglicky **Admission Control**), spadající do oblasti **předzpracování transakce**. Vstupní řízení zahrnuje techniky zabránění přetížení systému, tzn. chrání systémové zdroje RTDBS a minimalizuje pravděpodobnost, že transakce, která bude postoupena dalšímu zpracování se opozdí od časových omezení. Tyto situace nelze vždy eliminovat, pokud předem neznáme strukturu transakcí.

Modul **vstupního řízení** rozhoduje, zda-li bude transakce postoupena k dalšímu zpracování nebo v opačném případě bude zpracování transakce pozastaveno. Obecně můžeme specifikovat (18), že posláním vstupního řízení v kombinaci s algoritmy přidělování priorit je maximalizovat profit systému, což znamená, aby jednotlivé transakce byly vykonány s ohledem na jejich časové požadavky.

Základním cílem vstupního řízení, respektive mechanismů proti přetížení systému obecně, je **maximalizovat propustnost systému** při vysoké zátěži (13). Vstupní řízení tedy úzce souvisí s oblastí přidělováním priorit, což opět potvrzuje tezi, že je nezbytné databázové systémy reálného zkoumat jako celek.

2.4.1 Přidělování priorit

Transakcím je v RTDB systémech udělován přístup k procesoru na základě jejich priorit. Jedním ze základních konceptů RTDB systémů je sjednocení prioritně řízeného přístupu k procesoru a protokolů souběžného řízení pro dosažení maximálního stupně souběžnosti a využití zdrojů, samozřejmě při zachování konzistence dat (logické i dočasné), korektnosti zpracování a časových nároků transakcí. Z tohoto důvodu bylo nutné algoritmy přidělování priorit pro použití v RTDB systémech upravit.

V klasických real-time systémech je plánování úloh obvykle rozloženo mezi jednotlivé procesní komponenty jako CPU, I/O a plánovací procesy jsou navíc relativně nezávislé. Naplánovat zpracování transakce v RTDB systému však není jenom záležitostí přidělení priority pro přístup k procesoru, ale téměř v každé části transakčního zpracování musí být rovněž brán ohled na danou prioritu, především u mechanismu souběžného řízení a mechanismu obnovy dat. Rozvrhování priorit na kritických zdrojích systému musí být doplňováno prioritně založenou strategií zpracovávání transakcí na sdílených zdrojích (13).

Rozlišujeme **pevně a dynamicky přidělované priority** (24):

Fixně prioritní plánovač rozvrhuje úlohy přesně podle předem stanovených priorit. Na druhé straně dynamické plánování umožňuje měnit priority jednotlivých úloh během chodu systému. S ohledem na použitý algoritmus lze daným transakcím zvyšovat či snižovat prioritu pro zajištění jejich dokončení v určeném časovém termínu.

Pokud je transakce postoupena k dalšímu zpracování, tzn. je úspěšně zpracována modulem vstupní řízení, pak komponenta přidělování priorit dle daných algoritmů určí:

1. **Reálnou prioritu úlohy**, ve kterém bude transakce zpracována. Tato možnost nastane, pokud v systému platí, že každá transakce bude zpracována jako samostatně spustitelná jednotka (úloha) a systém zajistí paralelní zpracování. Pokud je na daném fyzickém počítači pouze jeden procesor, jde samozřejmě o **pseudoparalelismus** (28).
2. **Prioritu -** podle které bude transakce zařazena k dalšímu zpracování. Tento případ pokrývá tzv. **process pool**. V tomto případě není jednotlivá transakce spuštěna jako samostatná úloha, ale je vytvořeno více samostatných úloh a těm jsou jednotlivé transakce přidělovány ke zpracování.

Algoritmy plánování v prostředí reálného času rozdělujeme do dvou základních kategorií (4)**:**

1. **Statické algoritmy** vyžadují kompletní znalost o úloze a jejich omezeních, z čehož plyne jejich využití převážně pro oblast hard real-time systémů, tzn. nepoddajných RT systémů. Základní statické algoritmy přidělování priorit jsou detailněji popsány v (29).

2. V případě **dynamických systémů** neznáme dopředu množinu zpracovávaných úloh ani jejich časové požadavky, což může znamenat, že nové transakce mohou přijít do systému v neznámém čase.

Dynamické systémy lze dále dělit do dvou oblastí:

Systémy s dostatečnými zdroji (anglicky **Resource Sufficient Systems**) – jsou systémy, u nichž systémové zdroje garantují všem příchozím transakcím plánovatelnost. Opakem jsou pak systémy, u nichž nelze garantovat dostatečný rozsah systémových zdrojů. Dynamické algoritmy sice znamenají větší systémovou režii, ale mohou podávat lepší výsledky v závislosti na struktuře, typech transakcí a zatížení systému.

Velmi důležitá pro stanovení mechanismu přidělování priorit je rovněž periodicita transakcí, jestli jsou transakce **periodické** nebo **aperiodické**. V případě hard RTDBS se nejčastěji jedná o periodické transakce, které je možné plánovat výrazně lépe než náhodné transakce. Jednou z možností řešení je řazení transakcí podle přidělených priorit a preempce transakcí s nižší prioritou ve prospěch těch s vyšší prioritou. Avšak v případě velkého množství periodických transakcí s nízkou prioritou, které musí být odkládány z důvodu příchodu důležitějších aktivit, může dojít k totálnímu nedostatku zdrojů vyžadujících ukončení systému (13).

Pro stanovení priorit se kromě těchto vlastností používají i funkce, které berou v potaz real-time vlastnosti transakce a kombinují tak více informací dohromady (20). Jednou z možných kombinací charakteristik reálného času je kritický termín a kritičnost.

Priority mohou být v určitých případech navrženy předem, na základě měření, aby zajistily optimální činnost systému. Nejčastějším druhem měření je experimentální měření počtu transakcí, které promeškaly kritický termín. Toto měření je omezené pouze na transakce se stejnými prioritami.

Problém optimálního plánování transakcí je obecně výpočetně neřešitelný, a proto se musíme spokojit s heuristickými postupy.

2.4.2 Algoritmy přidělování priorit

Jako jedno z možných řešení pro předcházení trashingu[9] je zmíněná úprava algoritmů přidělovaní priorit, proto budou nejdříve popsány základní typy algoritmů přidělování priorit a jejich možné úpravy pro oblast databázových systémů reálného času. V této kapitole jsou popisovány protokoly pro plánování úloh[10] v jednoprocesorovém systému.

Základní typy algoritmů přidělování priorit

Deadline-Monotonic (zkratka **DM**) (18) algoritmus přiděluje priority transakcím na základě následujícího pravidla. Transakcím je přiřazena priorita na základě jejich kritického termínu, tzn. úloha s nejbližším kritickým termínem získá nejvyšší prioritu.

Rate-Monotonic (zkratka **RM** či **RMS – Rate Monotonic Scheduling**) (18), jedná se o speciální případ algoritmu DM pro periodické transakce, kdy za kritický termín považujeme právě periodu spouštění. Čím je perioda kratší (tzn. vyšší frekvence), tím vyšší je přidělená priorita. Tento algoritmus je optimální pro transakce přicházející periodicky, což nejčastěji nastává u hard databázových systémů reálného času. Výběr algoritmu plánování je však třeba u těchto typů systémů reálného času pečlivě zvažovat (29). Zajímavý analytický pohled lze nalézt v literatuře (24).

Poznatky o RMS algoritmu ukazují, že i při nekonečném počtu procesorů, jejichž **faktor využití procesoru** bude menší než 0.69, stačí použít fixně prioritní RMS. Obsáhla teorie o rozvrhování procesů má svou vypovídací hodnotu. Z aplikačního hlediska je důležité si uvědomit, že teoretické výsledky jsou odvozovány právě jen za určitých předpokladů, které v krajních případech nemusejí beze zbytku platit. Takovým předpokladem je například zanedbaní času na přepínání kontextu, což pro n s nízkou hodnotou může být přijatelné, ale pro $n \rightarrow \infty$ nikoliv.

Earliest Deadline First (zkratka **EDF**) (29) je nejčastěji používaný dynamický algoritmus. EDF algoritmus stanovuje nejvyšší prioritu transakci s nejbližším kritickým termínem. Algoritmus je optimálním dynamickým protokolem přidělování priorit při známém požadavku úloh na začátku zpracování. V rámci tohoto algoritmu jsou přednostně zpracovávány transakce s nejbližším absolutním kritickým termínem. Víme, že EDF je optimální algoritmus pro jednoprocesorový systém, viz důkaz v (18), strana 75.

Vytížení CPU při použití EDF algoritmu, lze zapsat následovně:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \quad \text{, kde}$$

C_i je doba trvání události, T_i je priorita události.

[9] „**Efekt trashingu**“ je popisován v kapitole č. 2.3.

[10] Bereme v potaz hledisko, že je transakce zpracovávaná samostatnou úlohou.

V kapitole č. 2.2.1 byly definovány dva druhy transakcí, přičemž transakce v jedné skupině mohou promeškat a v druhé skupině nemohou promeškat kritický termín. Transakce patřící do první skupiny lze plánovat pomocí EDF algoritmu.

Výkon algoritmu EDF však rapidně degraduje v situacích, kdy nastává přetížení systému.

Jak již bylo zmíněno v kapitole č. 2.4 o vstupním řízení transakcí, jedním z možných řešení přetížení je zavést do systému tzv. **ovladač zahlcení**, který definuje pravidla pro odmítání přicházejících transakcí, pokud zátěž systému dosáhne určité úrovně. Bez tohoto ovladače mají téměř všechny transakce v zatíženém systému dlouhé doby odezvy. Při použití ovladače zahlcení jsou alespoň některé transakce dokončeny v závislosti na jejich kritickém termínu a kritičnosti, dle použitého algoritmu.

Algoritmy přidělování priorit s ovladači zahlcení

AED algoritmus - varianta pro transakce se stejnou prioritou

Adaptive Earliest Deadline (zkratka **AED**) byl prezentován v (30). Algoritmus kombinuje **ovladač zahlcení** s **EDF** algoritmem a funguje následovně:

Po vstupu transakce ji systém vloží do fronty nevyřízených transakcí. Jestliže je její přiřazená pozice v intervalu 1..*H* (kde *H* je parametr), transakce je přiřazena skupině, která se nazývá HIT, zatímco zbývající jsou přiřazeny skupině MISS.

Cílem je pokusit se splnit všechny kritické termíny ze skupiny HIT, a pokud zbývá nějaký čas, provést transakce z MISS skupiny. Transakce v HIT skupině jsou řízeny EDF algoritmem. Transakce v MISS skupině jsou prováděny, jestliže v HIT skupině neexistují žádné nevyřízené transakce, v pořadí jejich umístění ve frontě. Jestliže je transakce jednou přiřazena určité skupině, zůstává v této skupině pořád. Po provedení transakce je tato odstraněna ze souboru nevyřízených transakcí.

H je řídící parametr pro daný RT systém. Jestliže se *H* blíží nekonečnu, systém se transformuje na EDF, jestliže *H* = 0, systém se transformuje na systém s náhodným výběrem transakcí. *H* bývá nastavováno podle následujícího algoritmu:

$$Úspěšnost\left(HIT\right)=\frac{Počet\ transakcí\ v\ HIT\ třídě\ se\ splněným\ kritickým\ termínem}{Celkový\ počet\ transakcí\ v\ třídě\ HIT}$$

$$Úspěšnost\left(ALL\right)=\frac{Počet\ transakcí\ se\ splněným\ kritickým\ termínem}{Celkový\ počet\ transakcí}$$

Když systém spustí měření, přiřadí prvních *H* transakcí skupině *Nh* a skupině N*a* všechny transakce, které přijdou. Potom jsou obě tyto skupiny sledovány, jak splní kritické termíny. Z tohoto měření se vypočte úspěšnost a provádí se opět nové měření. Počet transakcí *Nh* a N*a* musí být vhodně navržen. Systém po každém měření zvýší hodnotu *H* o 5 %, dokud není parametr Úspěšnost(ALL) po měření menší než 0.95. Pokud tedy Úspěšnost(ALL) klesne pod 0.95, *H* je opraven, snížen. Tyto hodnoty jsou v systému měřeny opakovaně.

Pomocí algoritmu lze posloupnost akcí zapsat následovně:

1. $H = Úspěšnost(HIT) * H * 1.05$
2. $IF\ Úspěšnost(ALL) < 0.95,\ potom$

   $H = min\ H,\ Úspěšnost(ALL) * Ntrans * 1.25,$

   kde Ntrans je celkový počet transakcí aktuálně prováděných v systému

Tento algoritmus zajistí, že pokud je v systému malý počet transakcí, většina transakcí je umístěna v HIT třídě. Pokud se blížíme k bodu, kde je Úspěšnost(ALL) menší než 0.95 (tj. více než 5% transakcí promeškalo svůj kritický termín), *H* je sníženo. Faktor 1.25 byl vybrán intuitivně. V porovnání s EDF algoritmy má AED algoritmus lepší výsledky pokud dochází k přetížení systému.

2.4.3 Mapování priorit

Dnešní operační systémy poskytují omezený rozsah priorit, na což poukazují metody pro mapování priorit (1), výkonnost systému a aplikace v reálném čase znatelně závisí na počtu úrovní priorit v operačním systému reálného času. Praktický návod, jaký rozsah priorit zvolit, lze nalézt v (31). Je důležité si uvědomit, že algoritmy souběžného řízení mohou být výkonnostně degradovány úzkým rozsahem priorit. V (2) je popsána souvislost mezi negativním efektem při mapování priorit u **Rate Monotonic** (zkratka **RM**) plánovacího algoritmu. Studie ukázaly, že při rozsahu 128 priorit je zde zanedbatelný výkonnostně degradační efekt. Při 64 prioritách máme 2% degradaci a při 32 prioritách degradace u algoritmu RM dosahuje 8%, při 8 prioritách dosahuje degradace 60%. Například u operačního systému VxWorks je 256 prioritních úrovní, 0 je nejvyšší prioritou. Pro uživatelské procesy je doporučen rozsah 100-255. Systémové procesy (úlohy) mají priority menší než 100. Platforma RTX má rozsah priorit 0-127, kdy 127 je nejvyšší priorita.

Z těchto informací je nutné vycházet při volbě rozsahu priorit pro vstupní transakce u databázového systému reálného času. Existují přístupy (32) obsahující pravidla pro mapování priorit na skutečné priority operačního systému, protože priority transakcí mohou být zvoleny v jiném intervalu, než který poskytuje platforma reálného času.

3. ZPŮSOB ŘEŠENÍ

Lze konstatovat, že bez existence testovací platformy databázového systému reálného času by nebylo možné navržené cíle práce realizovat a experimentálně ověřit v prostředí databázové systému reálného času. Proto autor textu intenzivně spolupracoval na vývoji experimentálního prostředí pro testování, neboli **experimentálního databázového systému reálného času**, dále nazývaného **Testbed**. Konkrétně se zabýval vývojem modulů **predispatcher**, **dispatcher** a portací existujících a integrací nově vyvinutých komponent systému Testbed na platformu RTX. Jednotlivé komponenty systému Testbed budou podrobněji popsány v kapitole č. 3.1.3.

3.1 EXPERIMENTÁLNÍ DATABÁZOVÝ SYSTÉM REÁLNÉHO ČASU

V roce 2005 vznikla výzkumná skupina, která od roku 2006 v rámci projektu Grantové agentury České Republiky (číslo projektu - 102/06/1742) pracovala s cílem vyvinout experimentální databázový systém reálného času. Skupina se skládala ze dvou aktivně spolupracujících týmů, přičemž první tým z Vysoké Škole Báňské Technické Univerzitě v Ostravě tvořily následující osoby: Doc. RNDr. Černohorský Jindřich, CSc.; Doc. RNDr. Šarmanová Jana, CSc.; Mgr. Jan Pokorný. Druhý tým byl složen z následujících členů: RNDr. Zdeněk Franěk; Ing. Václav Król ze Slezské Univerzity v Opavě – Obchodně podnikatelské fakulty v Karviné. Od druhé poloviny roku 2006 působili v týmu na Vysoké Škole Báňské Technické Univerzitě v Ostravě další členové, jmenovitě Ing. Vít Stratil, Ing. Ondřej Chlubna, Ing. Jan Kubíček, Bc. Tomáš Adámek.

Cíle projektu

Základním cílem skupiny bylo vyvinout modulární experimentální systém, na němž by bylo možno testovat nově vytvořené či modifikované algoritmy, přístupy či koncepty platné pro oblast databázových systémů reálného času. Dále, aby bylo možné na tomto systému testovat, jak významně se ovlivňují jednotlivé funkční bloky systému, závislost kombinací modulů na výkonnosti celého systému a počtu zpracovávaných transakcích a další významné systémové charakteristiky.

Při návrhu a vývoji systému Testbed byl kladen důraz na:

1) **Koncepční celistvost a jednoduchost návrhu jako celku**
   - pro další možnosti rozšíření či portaci na jiný operační systém reálného času

2) **Modifikovatelnost a flexibilitu**
   - možnost měnit jednotlivé části systému

3) **Minimalizaci režie systému**
   - při zachování možnosti detailního monitorování chování systému

Historie projektu

V prvním roce byla pozornost řešitelů soustředěna na analýzu problematiky a studium literatury, zabývající se především existujícími přístupy databázových systémů reálného času. Dále byly zkoumány základní mechanismy klasických databázových systémů a možnosti jejich adaptace do prostředí reálného času.

Byla provedena analýza koncepce systému Testbed (verze 1.0, s kódovým označením V4DB) a následně bylo navrženo základní schéma systému. Za vývojovou platformu byl zvolen operační systém reálného času VxWorks[11]. Během let 2005 a 2006 došlo k implementaci základních modulů systému a otestování systému jako celku. Průběžně publikované výsledky projektu, na kterých autor práce participoval, lze nalézt v kapitole tvůrčí publikace (1), (2), (3), (4), (5), (6). Testbed verze 1.0 byl popsán v disertační práci Václava Króla, (20).

Na základě dlouholetého a systematického přístupu k výzkumu se podařilo vymezit směry dalšího vývoje a proto byl navazující výzkum směřován do následujících oblastí:

- **Vstupní řízení databázových systémů reálného času**, touto oblastí se zabývá tato disertační práce.
- Příprava testovacích scénářů a provádění automatizovaných testů databázového systému reálného času, viz kapitola č. 5.1.
- Vytvoření matematického modelu databázového systému reálného času, viz kapitola č. 3.1.5.

3.1.1 Evoluce architektury systému Testbed

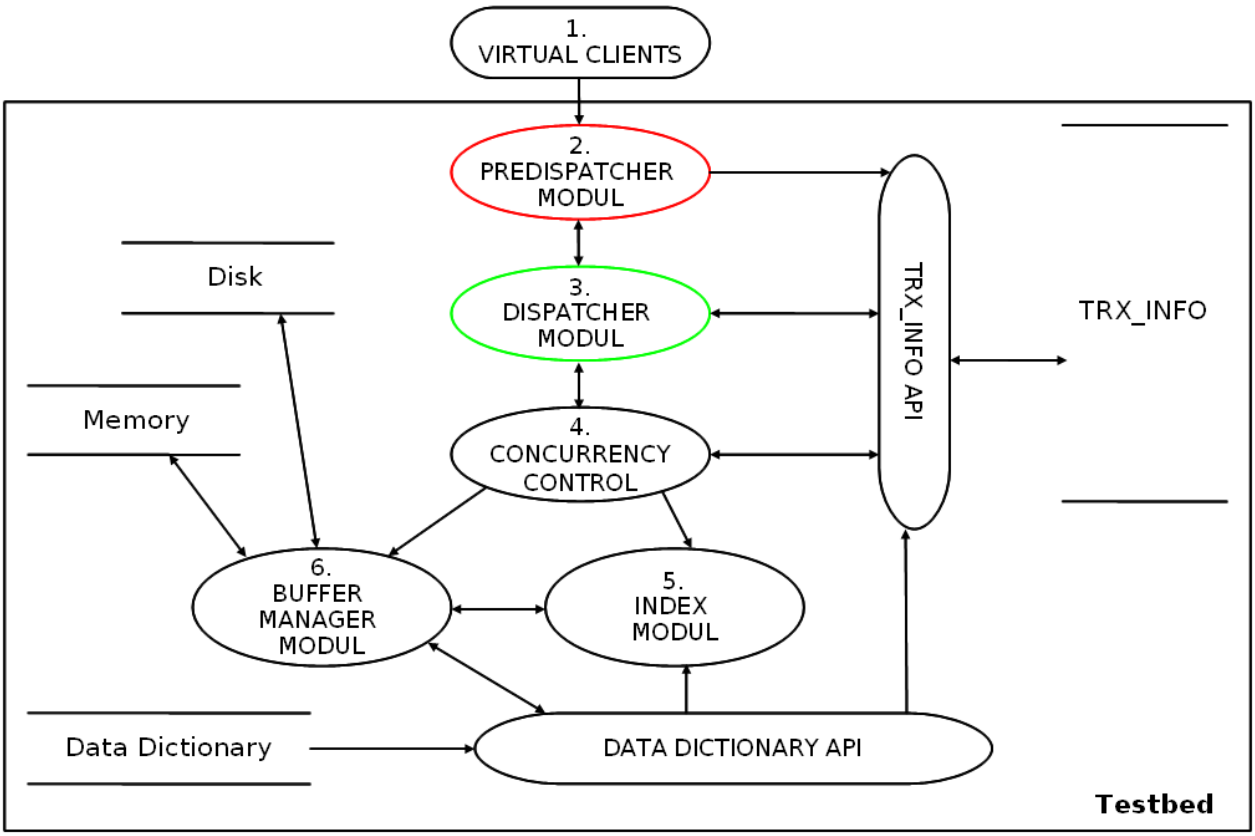
Z poznatků zjištěných během vývoje systému vyplynulo, že platforma VxWorks není vhodná pro další vývoj systému. Důvody vyplývají mimo jiné z porovnání s platformou RTX, které lze nalézt v kapitole č. 3.2.5. V následujících kapitolách bude popisován systém **Testbed verze 2.0**. S ohledem na další možné rozšiřování systému, integraci nově vzniklých modulů a lepší možnost testování systému byla upravena architektura předchozí verze systému a přidány nové komponenty, viz následující kapitola č. 3.1.2 popisující schéma systému Testbed.

Již od roku 2007 autor spolupracoval na nové verzi systému a jeho portování na novou platformu reálného času RTX, více v kapitole č. 3.2.3. Se změnou platformy reálného času došlo také ke změně programovacího jazyka, z jazyka C na C++. Aktualizovaná verze systému Testbed částečně využívá základní architekturu předcházejícího verze systému (20). Autor se přímo zabýval vývojem modulů **predispatcher**, **dispatcher sledování systému** (modul **Monitorovací Modul**) a přepracování kooperujících komponent, které komunikují s nově vzniklými moduly – **souběžné řízení** (33) a **resource manager**. Detailní popis systému lze nalézt v následující kapitole 3.1.2.Souběžně probíhal také vývoj nových modulů zajišťujících **indexování dat** (modul **Index Manager**) (34), **načítání dat** (modul **Buffer Manager**) (35).

[11] Více informací o tomto operačním systém reálného času lze nalézt v kapitole č. 3.2.2.

3.1.2 Schéma systému Testbed

V této kapitole si popíšeme schéma systému Testbed, a to z nejvyšší, tzv. nulté úrovně. Nultá úroveň data-flow diagramu[12] (zkratka DFD) popisuje systém Testbed v interakci s okolním prostředím a popisuje hlavní moduly systému na nejvyšší úrovni.



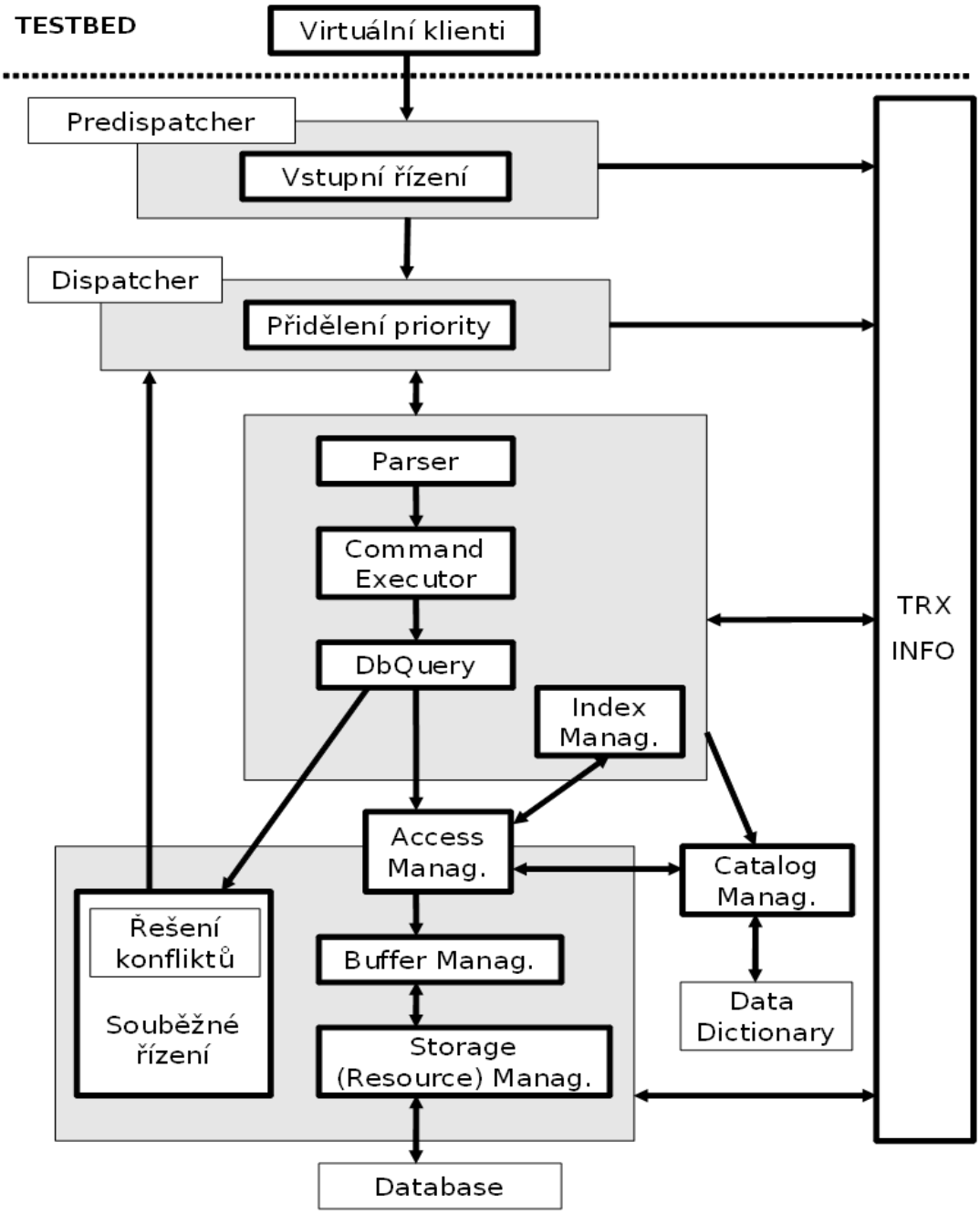
Obrázek 3.1: Schéma systému Testbed, DF diagram

Oproti systému v 1.0 přibyly moduly **Index Manager** a **Buffer Manager**, přičemž se logicky musel změnit i modul **souběžné řízení**. Přibyl systémový modul – Monitorovací Modul, viz kapitola č. 4.4.

Schéma první úrovně DF diagramu[13] nám systém Testbed přibližuje na nižší úrovni a popisuje funkční části jednotlivých modulů. Lze vidět interakce mezi jednotlivými moduly, jak konkrétně spolupracují s ostatními moduly systému Testbed.

[12] Obrázek úmyslně porušuje grafickou notaci data flow diagramu. Autor si je vědom tohoto porušení konvence, ale je přesvědčen, že toto doplnění poslouží k lepší čitelnosti a popisu schématu.

[13] Pro přehlednost je obr. 3.2 kombinací více úrovní DF diagramů do jednoho diagramu, přesto je zachována konvence pro DF diagramy. Snahou bylo dané schémata zobrazit co nejpřesněji pozměněním grafické notace pro specifický případ. Autor si je vědom, že takovému diagramu nelze plnohodnotně říkat data flow diagram.



Obrázek 3.2: Testbed - detailní schéma

Časovou posloupnost akcí při průchodu transakce systémem Testbed lze popsat v následujících krocích:

1. Transakce je na vstupu systému zpracována modulem **Predispatcher**, který transakci postoupí k dalšímu zpracování, pokud není systém přetížený. Informace o transakci jsou zaznamenány modulem **TRX_INFO**. Při průchodu každým modulem jsou informace o aktuálním stavu zpracování transakce uloženy, viz kapitola č. 3.1.3.

2. Transakci je následně přidělena priorita na základě RT charakteristik a zvoleného algoritmu v modulu **Dispatcher**. Dispatcher spustí zpracování transakce dle nastavení jako samostatné vlákno nebo zařadí do tzv. **Process Pool**, více v kapitole č. 2.4.1.
3. Dále je transakce rozdělena na jednotlivé databázové (tyto operace se dále dělí na čtecí a zapisovací) a systémové operace a začíná sekvenční zpracování jednotlivých operací transakce. O logicky správné paralelní zpracování transakcí se stará modul **souběžné řízení**, dle zvoleného algoritmu a dalšího nastavení.

3.1.3 Základní a systémové moduly

Tato kapitola obsahuje základní popis komponent systému Testbed. Detailní informace o modulech, které byly zaintegrovány do systému Testbed, což zahrnuje **Index Manager** (34), **Buffer Manager** (35) a modul **Souběžné Řízení** v (33).

Virtuální Klienti (anglicky **Virtual Clients**) se snaží co nejreálněji simulovat klienty RTDBS. Virtuální klienti jsou implementováni jako vlákna s časovačem pro periodické či aperiodické transakce. Pro odesílání transakcí v náhodných intervalech, byl využit speciální algoritmu generování náhodných čísel od tvůrců Karl-L. Noell a Helmut Weber. Tento algoritmus pro generování náhodných čísel je založena výsledcích z (36). Tyto transakce jsou posílány přes frontu zpráv do systému Testbed. Optimální varianta by byla generovat tyto transakce pomocí hardwaru, například speciální měřící karty. Bohužel jsou v tomto případě klienti softwarově simulovaní z důvodu jednodušší simulace a nastavení různých možností pro odesílání transakcí. Využití speciální měřící karty je doporučováno jako jeden z možných navazujících směrů do budoucna.

Transakce přicházející do systému musí nejprve projít mechanismem vstupního řízení, který monitoruje a reguluje celkový počet souběžně běžících transakcí v systému, s ohledem na zajištění RT charakteristik transakcí, především kritického termínu. Komponenta s názvem **Predispatcher** se stará o vyčítání transakcí pomocí fronty. Transakce se ukládá do sdílené paměti, kde se také vytváří struktura (TRX_INFO) uchovávající podrobnější informace o transakci[14]. Struktura TRX_INFO je využívána k vyhodnocení testů a monitorování celého systému.

Dispatcher se stará o **přidělování priorit** dle zvoleného algoritmu. V této části systému jsou určeny priority pro dané transakce a další způsob zpracování transakcí. S transakcí se pojí určité charakteristiky, které ovlivňují její prioritu. Mezi charakteristiky patří mimo jiné kritický termín a kritičnost viz kapitola č. 2.2.1. Pro stanovení priorit se kromě těchto vlastností používají i funkce, které berou v úvahu různé vlastnosti transakce a kombinují tak více vlivů najednou. Pro vykonání v této komponentě je důležité, zda je pro každou transakci vytvořena nová úloha či existuje množina úloh, ke kterých jsou jednotlivé transakce přiřazovány. Dispatcher přiděluje priority transakcím vstupujícím do systému, ale i transakcím, které jsou během svého zpracování restartovány.

V modulu dispatcher jsou implementovány následující algoritmy přidělování priority:

- **DM** a **RM** algoritmy pro aperiodické a periodické transakce

[14] Podrobný popis struktury TRX_INFO lze nalézt v příloze A.

- Algoritmus přidělování priorit transakcím, na základě jejich kritičnosti pro systém
- Speciální varianta algoritmu pro přidělování priority na základě kombinace kritického termínu a kritičnosti, přičemž lze procentuálně nastavit váhy jednotlivým parametrům, platí:

$$vd\left(\textit{váha kritického termínu}\right)+vc\left(\textit{váha kritičnosti}\right)=100\left[\%\right]$$

- Přidělování priorit naprosto náhodně, tento algoritmus byl vyvinut pro testování a vyhodnocení algoritmů přidělování priorit.

Transaction Manager (zkratka **TM**) se stará o zpracování transakce. Součástí modulu jsou následující části:

- **Parser** se stará o rozdělení samotných databázových operací a systémových příkazů v rámci transakce.
- **Command Executor** vykonává systémové příkazy a databázové operace předá ke zpracování modulu DBQuery. Následné zpracování samotných databázových operací probíhá v rámci jedné transakce vždy sekvenčně.
- **DBQuery** zpracovává jednotlivé databázové příkazy na logické úrovni.

Indexovací modul (anglický ekvivalent **Indexing Manager**, zkratka **IM**) umožňuje přistupovat a testovat způsoby přístupu k indexovaným datům. Lze testovat tři možné způsoby indexování: indexaci dat v paměti, na disku a nebo indexaci dat na disku i v paměti ve spolupráci s modulem pro práci s daty – **Storage Manager**.

Modul pro práci s daty (anglický ekvivalent **Buffer Manager**, zkratka **BM).** Nelze vyloučit případy, kdy nelze všechna požadovaná data uložit do paměti, proto byl vyvinut modul **buffer manager**, který umožňuje ukládání aktuálně nepotřebných dat na pevný disk. Modul se transparentně stará o práci s daty, ať už jsou uložena přímo v paměti nebo na disku.

Souběžné řízení (anglický ekvivalent **Concurrency Control**) je velmi důležitá část databázového systému, která se stará o logicky správné paralelní zpracování transakcí. Pro zajištění souběžného řízení se používají protokoly založené na zamykání dat, optimistické protokoly, spekulativní protokoly či jiné. Tento modul, který obaluje jednotlivé elementární operace potřebnými zámky, kontroluje konflikty mezi transakcemi a řeší vzniklé problémy pomocí zvyšování priorit. Zajišťuje restart transakcí, pokud je jisté, že nestihnou svůj kritický termín vykonání. Tento modul má několik vstupů i výstupů pro komunikaci s ostatními moduly. Modul **souběžného řízení** komunikuje s modulem **indexování**, **buffer managerem** a také s modulem **dispatcher**.

Data Dictionary (datový slovník) vytvoří podle definice tabulek a atributů strukturu pro uchovávání dat.

Storage Manager se stará o přístup k datům uloženým na fyzickém médiu, tzn. souborový přístup, pokud jsou data uložena na pevném disku.

Systémové moduly

Monitorovací modul zaznamenává informace o aktuální stavu systému, modul je popsán v samostatné kapitole č. 4.4.

Informace o transakcích během zpracování se ukládají do struktury TRX_INFO. Údaje ze struktury TRX_INFO jsou po ukončení běhu systému Testbed exportovány do souboru typu csv. Explicitně známá struktura umožňuje další vyhodnocení a automatické zpracování daných výsledků při běhu systému.

TRX_INFO

Struktura TRX_INFO pro systém Testbed byla rozšířena o více parametrů[15], které jsou detailněji popsány v příloze A. Prvotní naplnění obsahu struktury pro danou transakci má za úkol modul **Predispatcher**. Ostatní součásti systému ovšem do této oblasti přistupují a vyčítají definici transakce nebo její real-time charakteristiky a zároveň zapisují procesní údaje během jejího zpracování, především časové značky počátku a ukončení jednotlivých fází zpracování nebo informaci o tom, zda byla transakce potvrzena (operace COMMIT) či byla ukončena bez potvrzení (operace ABORT). Rovněž se eviduje počet restartů a počet blokací transakce.

Modul **TRX INFO** poskytuje přístup ke struktuře uložené v paměti, která je vytvářena během inicializace systému pro určitý pevný počet transakcí. Tento počet je definován systémovou konstantou, která tím vlastně určuje maximální počet transakcí, které mohou být systémem zpracovány. Z hlediska variability systému by jistě bylo lepší vytvářet danou strukturu dynamicky pro každou transakci při jejím vstupu do systému. Avšak vzhledem k tomu, že alokace paměti je časově relativně náročnou operací v rámci operačního systému, bylo rozhodnuto o perzistentním vytvoření této struktury před spuštěním systému. Tento způsob je rovněž vhodný z toho důvodu, že při provádění jednotlivých experimentů je počet vstupních transakcí znám předem a experimenty tak nejsou zatíženy režií systémových operací.

Detailní přehled o jednotlivých parametrech udává tabulka v příloze A. Tento přehled je vhodné uvést vzhledem k důležitosti popisované struktury a pro doplnění předchozích informací.

Zobrazovat informace o systému během zpracování, mimo debuggovací režim, nemá smysl z důvodu časové režie výpisu informací na obrazovku. Po ukončení chodu systému lze informace o chodu systému získat ze souboru Trx_Info.csv. Tento soubor obsahuje informace získávané modulem TRX_INFO během chodu systému, které jsou vhodně strukturovány pro další zpracování, viz příloha A.

[15] V porovnání s experimentálním systémem reálného času - Testbed verze 1.0.

Základní parametry ukládané systémem, který charakterizují transakci:

- **čas příchodu** a_i: čas, kdy byla transakce přijata systémem
- **výpočetní čas** C_i: čas, po který byla transakce zpracovávána
- **deadline** d_i: čas, do kdy by měla být transakce úspěšně zpracována
- **čas spuštění** s_i**:** čas začátku zpracování transakce
- **čas ukončení** f_i: čas ukončení zpracování transakce

a další parametry viz příloha A.

Logovací subsystém

Tuto část systému není třeba detailně rozebírat. Je ovšem vhodné její existenci zmínit, neboť logování jednotlivých akcí výrazně usnadnilo ladění experimentálního systému Testbed. Informace, které jsou prováděny lze vypisovat podle nastavení:

- na obrazovku
- do souboru

Tento subsystém je implementován pomocí podmíněné kompilace[16] a během experimentálních běhů systému je možné tento subsystém vzhledem k jeho časové režii vypnout. Logovací subsystém poskytuje různé úrovně nastavení, které lze zaznamenávat ALERT, CRITICAL, ERROR, WARNING, NOTICE, INFO, DEBUG, TRACE.

Popisovaný modul **TRX INFO** je zobrazeny na schématu č. 3.1.2, **logovací** subsystém nikoliv, a to z důvodu, že není nezbytný pro chod systému a je součástí systému pouze, pokud explicitně povolíme možnost logování. Tento mechanismus je zajištěn pomocí podmíněné kompilace programovacího jazyka.

3.1.4 Konfigurace systému

Konfigurace systému Testbed a jeho jednotlivých komponent je uložena v pěti konfiguračních souborech **cAppconst.h, cDB.h, startup.ini**, **db.ini** a **clients.ini**.

První soubor **cAppconst.h** obsahuje:

- cesty ke konfiguračním souborům
- definice systémových proměnných a datových struktur
- nastavení debuggování pro jednotlivé komponenty systému
- nastavení testovacích konstant
- nastavení jednotlivých modulů

[16] SOBOL, Tomáš. *Výuka C++ Builder*. 2001 [cit. 2011-12-15]. Dostupné z: http://builder.tsx.cz/1/hl.html.

Soubor **cDB.h** popisuje:

- definici proměnných a datových struktur souvisejících se zpracováním databázových operací

Soubor **startup.ini** definuje:

- nastavení modulů systému
- priority, typy algoritmů a další nastavení modulů systému

Soubor **db.ini** obsahuje:

- definici tabulek a jejich atributů

```
create table tab1 (
 atr1 string index;
 atr2 string;
 atr3 short int;
 atr4 int;
 atr5 double;
);

tablesize = 100;
```

Obrázek 3.3: Definice tabulky

Soubor **clients.ini** obsahuje nastavení pro jednotlivé klienty:

- informace o počtu klientů a jejich identifikátor
- nastavení priorit
- jestli budou do systému transakce posílány periodicky či aperiodicky. V případě aperiodicity interval pro náhodné odesílání transakcí. V případě periodicky generovaných transakcí, pak obsahuje informaci o periodě.
- počet, kolikrát budou jednotlivé transakce odeslány do systému

Jeden klient může poslat v definovaných interval i více transakcí. Definice transakcí pro jednotlivé klienty lze nalézt v souborech **lc0.trx** až **lc**[*X*]**.trx**, kde parametr *X* určuje počet klientů. Každý soubor obsahuje definici transakcí pro právě jednoho klienta.

3.1.5 Verifikace systému

V oblasti počítačových systémů verifikace dokazuje nebo vyvrací správnost systému vzhledem k dané formální specifikaci použitím matematických formálních metod[17]. Verifikace popisovaného systému Testbed pomocí matematického modelu či nástrojů je důležitá, ale také velmi složitá. S čím se během verifikace tak komplexního nástroje jako je systém Testbed, setkáme a co lze také dnešními nástroji verifikovat pomocí matematických modelů, zejména v nástroji **Uppaal**[18], se zabýval jeden z členů vývojového týmu Martin Kot, který prvotní výsledky výzkumu představil v publikaci (37). Odkazovaná publikace představuje matematický model pro část souběžného řízení systému Testbed. Další možností, jak systém verifikovat, je použití alternativního postupu, například pomocí verifikačního nástroje **RT-SPIN**[19] či jiných nástrojů.

3.2 RUNTIME A VÝVOJOVÉ PROSTŘEDÍ

Lze jedině souhlasit s obecným přístupem, že dokonalejší garance činnosti systémů reálného času za nejrůznějších podmínek, lze dosáhnout pouze tehdy, pokud bude použito dokonalejších návrhových postupů v kombinaci se statickou, dynamickou analýzou kódu a s využitím specifických mechanismů operačního systému účelově navržených pro podporu systému v omezujících časových podmínkách. Následující kapitola klade důraz na požadavky systému Testbed z implementačního hlediska.

Další kapitoly proto přinášejí popis platforem reálného času, na kterých byl systém vyvíjen. Již na začátku vývoje systému Testbed bylo jasné, že jde o rozsáhlý softwarový projekt, pro jehož existenci není nezbytně nutná existence speciálního hardware, což může být však cílem navazujícího projektu. Dále je popisováno s jakými překážkami se lze při vývoji softwarové aplikace reálného času u daných RT platforem setkat a čeho jsme se vyvarovali během vývoje systému Testbed.

3.2.1 Požadavky kladené na systém Testbed

Požadavky na systém Testbed jsou v souladu s obecnými požadavky na kritický RT systém, respektive na aplikace reálného času. Při vývoji těchto typů aplikací preferujeme následující vlastnosti (24):

- **časová přípustnost** – výsledky musejí být platné v čase jejich získání
- **návrh postupu při výskytu špičkových přetížení** – systém se nesmí zhroutit, pokud je podroben přetížení, musí být opatřen mechanismem pro eliminování těchto situací

[17] Harkonen, Janne., ed. Maturity of verification and validation in ICT companies. In: *International Journal of Innovation and Learning 2009*. Vol. 6, 2009, s. 33 - 50.

[18] *Uppaal V4™ The verification platform* [software]. [přístup 15. prosince 2011]. Dostupné z: http://www.uppaal.com/.

[19] *REAL-TIME SPIN* [software]. [přístup 15. prosince 2011]. Dostupné z: http://www-verimag.imag.fr/~tripakis/rtspin.html.

Základní požadavky při vývoji byly také kladeny na možnost efektivního testování jednotlivých částí kódu. Při vývoji a implementaci systému Testbed na novou platformu RTX byl z programátorské hlediska kladen důraz na následující tři aspekty:

1. **Efektivita**
2. **Bezpečnost**
3. **Modifikovatelnost**

Efektivita

Při vývoji se zaměřujeme na **efektivitu**, což znamená, že změna programovacího jazyka nesměla způsobit výsledné zpomalení chodu systému Testbed a muselo být zaručeno, že nedojde k výraznému prodloužení délky zdrojových textů. Jak již bylo řečeno Testbed verze 2.0 byl naprogramován v jazyce C++. „Pasti" jazyka C++ lze omezit vhodným návrhem architektury programu a s využitím výhod C++ lze získat kód minimálně stejně efektivní jako při použití jazyka C, viz analýza výkonosti C versus C++ [20]. Další výhodou jazyka C++ je přísnější typová kontrola. Konstrukce kódu vytvořeného a zkompilovaného v jazyce C, nemusí jít jednoduše a bez chyb zkompilovat kompilátorem pro jazyk C++, přesto lze volat části kódu jazyka C v jazyce C++.

Požadavky na maximální výkonnost při běhu aplikace v jazyce C++ a jejich splnění při vývoji systému Testbed jsou zajištěny mimo jiné následujícím seznamem doporučení, který byl při vývoji systému dodržován:

- nepoužívání standardních šablon jazyka C++
- optimalizace práce s pamětí
- omezení používání systémových knihoven, z důvodu případné optimalizace či nutnosti změn knihoven na nejnižších úrovních
- nepoužívání výjimek
- používání deterministických funkcí, které poskytuje platforma reálného času[21]

Při vývoji systému byly využity výhody jazyka C++, které přispěly ke zpřehlednění kódu, přičemž bylo využito objektového návrhu. Na druhou stranu nejsou využívány výjimky, které jsou sice velmi elegantní, ale nedeterministické. Celý tento přístup přinesl výhodu čitelnějšího, snadněji udržovatelného a znovupoužitelného zdrojového kódu pro další možnosti rozšiřování a testování.

Bezpečnost

Bezpečnost systému Testbed ve smyslu správného chodu byla zajištěna splněním následujících bodů (38):

- správné nastavení kompilátoru
- explicitní přetypování všech proměnných

[20] Østergaard, Jakob. *C versus C++* [online]. 2004 [cit. 2011-12-15]. Dostupné z: http://unthought.net/c++/c_vs_c++.html.

[21] V dokumentaci k produktu RTX lze najít seznam funkcí, které jsou deterministické.

- možnost detekovat a především správně zobrazovat chyby pomocí logovacího modulu systému. Bohužel nebylo možné použít jedno z mnoha existujících řešení např.: **log4cpp**[22], a to z důvodu náročnosti a možnosti ovlivnění výkonnosti systému jako celku. Proto byl vyvinut vlastní logovací subsystém, sloužící k zaznamenávání informací o běhu systému. Logovací modul je zakomponován do systému s využitím podmíněné kompilace. Tato skutečnost znamená, že pokud ho nebudeme chtít využívat, nebudou jeho volání zakompilována do výsledného programu a tento modul nebude ovlivňovat celkovou výkonost systému Testbed.

Udržovatelnost a modifikovatelnost

Systém Testbed byl implementován v jazyce C++, který umožňuje modifikovatelnost a snadnou rozšiřovatelnost systému. Tyto požadavky není úplně jednoduché splnit. Pro podporu a do jisté míry také pro kontrolu během vývoje aplikace, a také ke kontrole celkového návrhu, nám sloužil softwarový nástroj software **CCCC**[23], vyhodnocující softwarové metriky a vypisující informace pro následující tři typy metrik:

- Procedurální metriky, měření na úrovni jednotlivých funkcí: počet řádků kódu, počet řádků komentářů a další, tzv. McCabe's Cyclomatic Complexity (39)
- Metriky objektového návrhu: počet úrovní dědění, počet metod na třídu a další parametry
- Strukturální metriky založeny na práci Henry & Kafura (40): fan-in, fan-out, information flow

3.2.2 Operační systém reálného času VxWorks

Pro vývoj první verze systému Testbed (s kódovým jménem V4DB (20)), byl zvolen operační systém reálného času **VxWorks**[24], vyvíjený firmou WindRiver[25]. VxWorks patří mezi velmi rozšířený operační systém reálného času, zejména v oblasti průmyslových aplikací. Vzhledem k zaměření této práce je dále uveden pouze stručný popis základních vlastností, které tento operační systém charakterizují a je vhodné je uvést před dalším textem popisujícím platformu **RTX**, viz následující kapitola č. 3.2.3.

Základem operačního systému VxWorks je výkonné mikrojádro Wind. Mikrojádro zahrnuje nástroje pro podporu reálného času a poskytuje následující vlastnosti:

- rychlý **víceúlohový režim** (anglicky **Multitasking**), neomezený počet úloh
- **preemptivní** a **cyklické plánování**
- rychlé deterministické **přepínání kontextu** (anglicky **Context Switching**)
- rychlý a výkonný přerušovací systém a chybové rutiny
- **komunikace mezi procesy** (anglicky **Inter-Process Communication**, zkratka **IPC**)

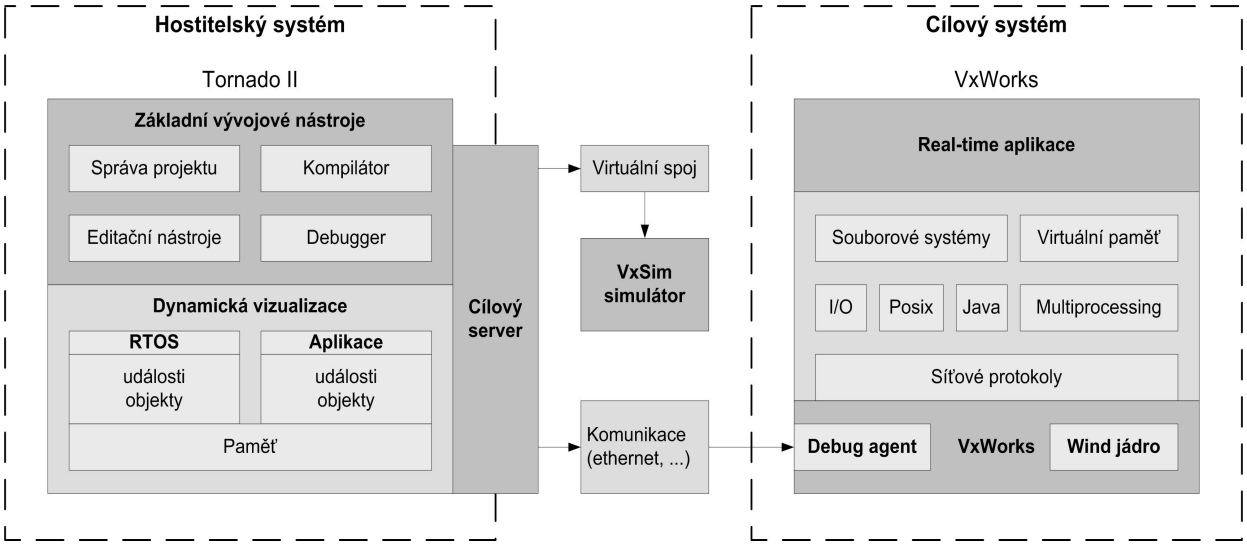
[22] *Log for C++* [software]. [přístup 15. prosince 2011]. Dostupné z: http://log4cpp.sourceforge.net/. Požadavky na systém: Neuvedeno

[23] TIM LITTLEFAIR. *CCCC - C and C++ Code Counter* [software]. [přístup 15. prosince 2011]. Dostupné z: http://cccc.sourceforge.net/.

[24] Wind River. Wind River VxWorks. *www.windriver.com* [online]. [cit. 2011-12-15]. Dostupné z: http://www.windriver.com/products/vxworks/.

[25] Wind River. About Wind River. *www.windriver.com* [online]. [cit. 2011-12-15]. Dostupné z: http://www.windriver.com/company/.

- binární a čítací **mutexy, semafory** s prioritní dědičností
- zaznamenávání událostí pomocí **žurnální služby** (anglicky **Message Logging**)
- lokální a distribuované **fronty zpráv** (anglicky **Message Queue**)
- **sdílená paměť** (anglicky **Flat Memory model**), tzn. všechny procesy mají stejná práva pro přístup do paměti.



Obrázek 3.4: Vývojové prostředí Tornado II

Řízení procesů

Operační systém VxWorks umožňuje dva způsoby plánování procesů. Standardně je nastaveno preemptivní **prioritní plánování** (anglicky **priority-based**), které lze programově měnit na **cyklické plánování** (anglicky **round-robin**). Periodická vlákna nejsou k dispozici a je nezbytné používat časovače. Nastavit je možné 256 prioritních úrovní v rozsahu 0–255. Číslice 0 vyjadřuje nejvyšší prioritu, pro aplikační úlohy se doporučuje využívat hodnoty priorit od 100–255. Počet vláken je omezen pouze rozsahem dostupné paměti.

Paměťový model

VxWorks nemá žádnou explicitní ochranu mezi uživatelským a systémovým paměťovým prostorem. Tato skutečnost platí pro verzi 5.5, pro VxWorks verzi 6 jsou již možnosti trochu jiné. Experimentální systém Testbed verze 1.0 byl ovšem implementován na v té době dostupnou verzi 5.5. Bylo využito možnosti, kdy všechny procesy mají neomezený přístup k celému paměťovému prostoru, tzv. paměťový model **Flat Memory**.

Kompatibilita

Přenositelnost aplikací na jiný RTOS je omezená. Jsou zde k dispozici dvě aplikační rozhraní:

- proprietární rozhraní
- rozhraní odpovídající standardu POSIX 1003.1b

Rozhraní dle standardu POSIX 1003.1b nebylo pro verzi VxWorks 5.5 plně implementováno. To je hlavním důvodem, proč nebylo toto rozhraní při vývoji systému Testbed verze 1.0 využito (20).

Vývojové prostředí Tornado II

VxWorks tvoří vlastně runtime komponentu prostředí Tornado II, které je díky tomu tedy nejvhodnějším nástrojem pro implementaci softwarových systémů postavených nad tímto OS. Prostředí zahrnuje komplexní sadu křížových vývojových nástrojů, což umožňuje vývoj aplikací s minimálním podílem práce v cílovém systému. To je jednou z jeho hlavních předností. Vývoj probíhá na hostitelském systému, aniž by bylo nutné mít k dispozici cílový hardware, na kterém bude vyvíjená aplikace nasazena. Tyto možnosti dále rozšiřuje integrovaný simulátor VxSim, který disponuje vlastní časovou základnou a chování aplikací lze vyzkoušet pomocí tohoto simulátoru před nasazením na cílový systém. Navíc lze využít softwarového logického analyzátoru WindView, který umožňuje přehledné grafické zobrazení všech důležitých run-time údajů o průběhu aplikace, např. událostí týkajících se přepínání kontextu, zasílání a čtení zpráv z front, nastavení signálů, atd. Aplikace lze vyvíjet v jazyce C nebo C++. Obrázek č. 3.4. zobrazuje vztahy mezi nejdůležitějšími prvky vývojového prostředí Tornado v rámci hostitelského systému a cílovým systémem. Komunikace mezi hostitelskými nástroji a systémem VxWorks je zprostředkována přes cílový server a cílového agenta, schéma je převzato z (20).

3.2.3 Platforma reálného času RT eXtension

Platforma **RTX,** což je zkratka pro **Real Time eXtension**[26], vyvíjená firmou IntervalZero (dříve Ardence) není klasickým operačním systémem reálného času. Funguje jako rozšíření operačních systémů rodiny Microsoft Windows. Jako zdroje následujících informací byly použity zdroje (41), (42).

Platforma RTX již od verze 6.5 dokáže spolupracovat s následujícími verzemi operačního systému firmy Microsoft:

Verze OS	Počet CPU	Aktualizace
Windows 2000 Professional	1-4 CPU	SP4
Windows 2000 Server	1-4 CPU	SP4
Windows XP Professional	1-2 CPU	SP2
Windows XP Embedded	1-2 CPU	SP2

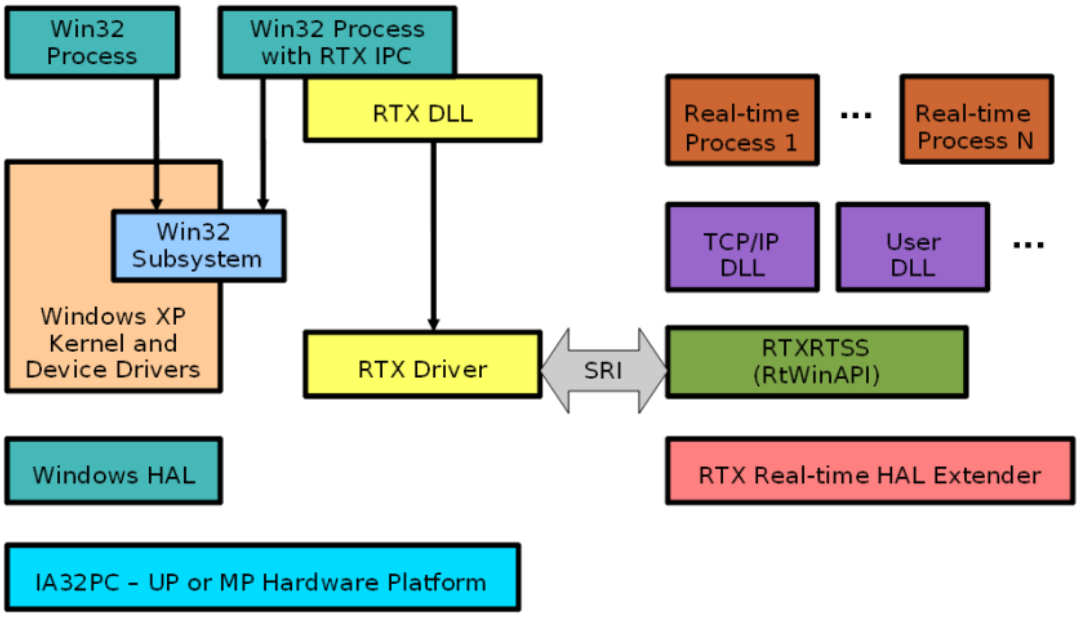
Tabulka 3.1: Typy MS Windows podporované platformou RTX

Typické nasazení tohoto prostředí je pro aplikace, ve kterých je kladen požadavek na sběr dat v reálném čase z určité periférie, např. ze sítě, přičemž bezprostředně poté následuje požadavek, tato data graficky zobrazit, případně dále zpracovávat v prostředí operačního systému MS Windows.

[26] INTERVALZERO. *RT eXtension - RTX* [software]. [přístup 15. prosince 2011]. Dostupné z: http://www.intervalzero.com/products/rtx-downloads/.

Architektura

RTX je rozšíření, které nepřekáží běhu ani nemodifikuje infrastrukturu systému Windows. Tímto je zajištěno, že RTX aplikace běží nezávisle během a také po případné havárii systému Windows. Následující obrázek byl převzat a upraven z webových stránek společnosti Microsoft[27].



Obrázek 3.5: Architektura RTX

Základní vlastnosti RTX:

- Obsluha přerušení na bázi zpráv, pro zařízení kompatibilní s MSI/MSI-X, poskytuje alternativu k přímé obsluze přerušení
- Podpora funkcionalit MMX a SSE/SSE2/SSE3
- V časovači **RTX HAL podpora kratších časových period než 1 µs**
- Podpora komunikace Firewire a USB, **jádro ladění využívá WinDbg**
- Vysokorychlostní mechanismus komunikace
- Podpora **zamezení inverze priorit**
- Rychlost obsluhy **přerušení do 30 kHz**

Řízení procesů

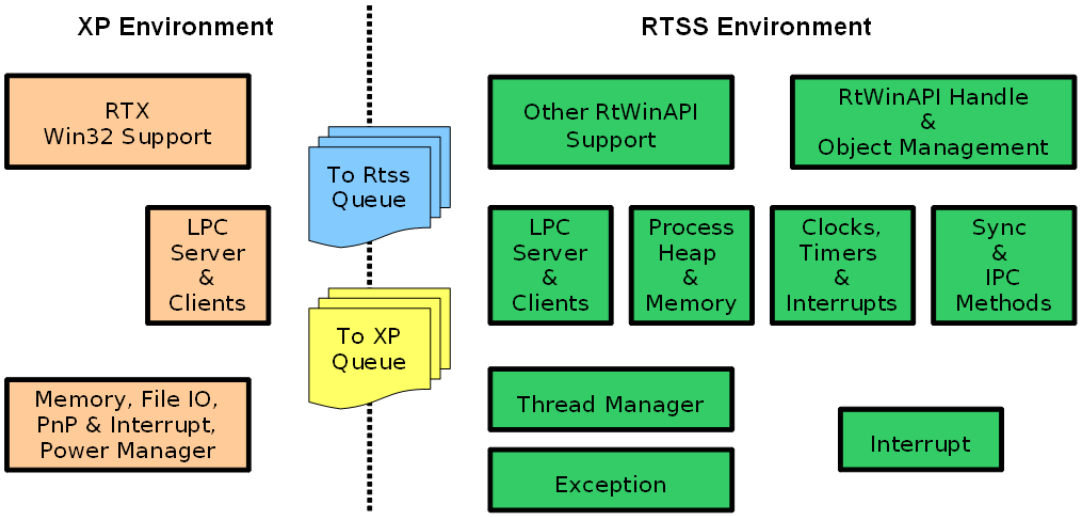
Jádro RTX RTSS je navrženo nad vysokorychlostním plánovačem, který podporuje oba algoritmy **preemptivní** a **round-robin**. RTX podporuje až 1000 nezávislých procesů a neomezený počet programových vláken.

---

[27] VENTURCOM. Hard Real-Time with Venturcom RTX on Microsoft Windows XP and Windows XP Embedded. *http://msdn.microsoft.com* [online]. 2003 [cit. 2011-12-15]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms838583%28WinEmbedded.5%29.aspx.

IPC mezi operačním systémem MS Windows a RTX

Výhodou rozšíření RTX je možnost zpracovávat získaná data v prostředí platformy Windows.



Obrázek 3.6: Platforma RTX z aplikačního pohledu

Zpětná kompatibilita a přenositelnost

Programování pro platformu RTX je ze své podstaty propojené s platformou MS Windows a tedy nepřenositelné. RTX platforma je rozšíření win32 operačního systému o reálný čas. Maximálně máme možnost zkompilovat aplikaci na win32 platformu. Na druhou stranu RTX API poskytuje množství funkcí podobných s API OS MS Windows, pouze rozšířené o prefix RT. RTX API je plně kompatibilní s Win32 API, tzn. není potřeba používat wrapování kódu pro mapování API. V dokumentaci produktu RTX jsou detailně popsány funkce, které nejsou deterministické a při programování pro aplikace reálného času by se proto měly používat s rozvahou.

Paměťový model

Již ze základního popisu platformy RTX lze vidět, že aplikace mohou jednoduše sdílet paměť naplněnou RTX aplikacemi, tzn. existuje sdílená paměť s win32 procesy.

3.2.4 Validace platformy RTX

Reálné chování platformy RTX bylo zkoumáno a porovnání s platformou RT-Linux provedeno spolupracovníkem vývojového týmu, Pavlem Morycem pomocí vyvinutého systému **RT-Golem** (43). Při testech byla sledována a měřena přesnost plánovače, latence přerušení, doba přepnutí vláken, latence semaforu, doba zamknutí a odemknutí mutexu a další systémové parametry. Výsledky těchto experimentů jsou podrobně popisované v (44).

3.2.5 Platformy RTX a VxWorks z pohledu vývoje

Autor měl možnost pracovat s následujícími verzemi softwarových platforem reálného času:

- VxWorks 5.5
- RTX 7.0 a RTX 8.1

Dále proto budou popisovány výhody a nevýhody pouze zmiňovaných verzí daných produktů, a to především z pohledu vývojáře a také test inženýra. Je samozřejmě nevhodné porovnávat verze softwaru, které byly vyvinuty v rozmezí 3 let, přesto základní koncepty obou platforem jsou platné již delší dobu a lze předpoklad, že ještě nějakou dobu platné zůstanou.

Z vývojářského hlediska musíme vyzvednout velkou výhodu platformy RTX, neboť k vývoji aplikací pro tuto platformu lze využít nástrojů, které jsou známé pro platformu Windows. Konkrétně u nástroje Microsoft Visual Studio[28] lze s výhodou využít přístupů pro práci se známým vývojovým nástrojem. Obdobným směrem se vydala také firma WindRiver, která za vývojový nástroj pro novější verze VxWorks zvolila platformu Eclipse[29], místo vývojového nástroje Tornado II. Velkou výhodou, nejen pro testování, při vývoji na platformě RTX je zobrazení informací na úrovni platformy Windows.

Z pohledu testera lze pomocí zabudovaného debuggeru odchytávat chyby při spouštění debug verze RTSS aplikace v systémové úrovni RING 3. Lze také využít známého krokování. Na druhou stranu je pro RTX platformu skoro nemožné najít a využít existující zdrojové kódy, či ukázkové příklady, mimo těch, které jsou součástí instalačního balíku. Další nevýhodou je také malé množství diskuzních skupin zabývajících se vývojem pro platformu RTX.

Nástroje pro vývoj RTX aplikací:

1. **ActiveView** – sleduje 10 interních API funkcí RTX v reálném čase.
2. **TimeView** – zobrazuje interakce mezi procesy a použitými vlákny uvnitř RTX.
3. **KernelView** – umožňuje „živé“ odlaďování v jádře, je součástí WinDbg.
4. **PerformanceView** – monitoruje využití CPU, jak ze strany RTX tak Windows.

Během práce s platformou RTX se objevily také nedostatky. Nefungovala např. funkce *watch*, sledování hodnot v debuggovacím režimu. Bohužel v mnohých případech nelze zobrazit konkrétní proměnné náležící jednotlivým třídám a z toho důvodu je nutné si dané hodnoty proměnných vypisovat. Pomocí RTX utility lze zobrazit pouze hodnoty lokální proměnné.

Přes všechny nedostatky prostředí RTX lze však konstatovat, že vývoj aplikace pro RTOS RTX je mnohem efektivnější v porovnání s vývojem na platformu VxWorks. Změna cílové platformy významně usnadnila vývoj a testování systému Testbed.

---

[28] MICROSOFT. *Microsoft Visual Studio* [software]. [přistup 15. prosince 2011]. Dostupné z: http://www.microsoft.com/visualstudio/en-us/.

[29] ECLIPSE FOUNDATION. *Eclipse IDE* [software]. [přístup 15. prosince 2011]. Dostupné z: http://www.eclipse.org/org/.

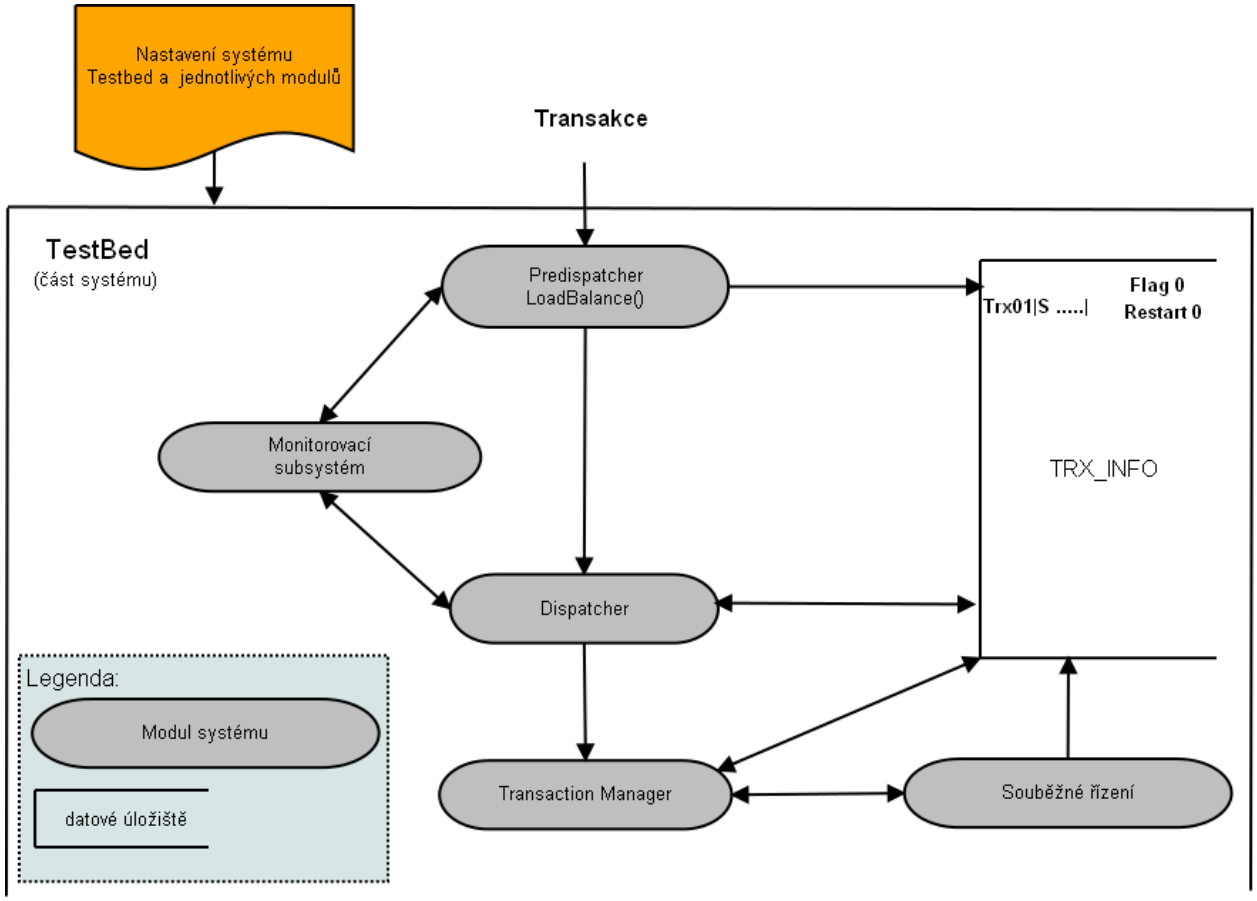
4. POSTUP ŘEŠENÍ

Navržené cíle práce byly řešeny v následujících etapách:

- **Analýza, návrh** a **vývoj experimentálního systému Testbed**, podrobnější popis lze nalézt v kapitole č 3.1.
- **Analýza**, **návrh** a **implementace algoritmu** pro **vstupní řízení** transakcí **založeného na zpětné vazbě systému,** kapitola č. 4.1.
- **Navržení vhodných váhových funkcí** pro vyhodnocení testovacích případů, kapitola č. 5.3.
- **Návrh** a **vývoj monitorovacího modulu**, který se stará o sledování chodu daného systému, zjišťuje zatížení systému Testbed, více v kapitole č. 4.4.
- Testování různého nastavení systému a vstupních transakcí a **sledování situací, kdy dochází k přetížení systému**, kapitola č. 5.5.
- **Vyhodnocení algoritmu vstupního řízení** v kooperaci s monitorovacím modulem a při vhodně zvolených váhových funkcích otestování, kapitola č. 5.

4.1 ANALÝZA A NÁVRH VSTUPNÍHO ŘÍZENÍ

Následující kapitola popisuje součásti algoritmu vstupního řízení, který je založen na informacích o aktuálním stavu systému, tzn. je založen na zpětné vazbě. Cílem bylo nalézt co nejefektivnější řešení proti přetížení systému Testbed, u něhož není dopředu známa struktura a počet vstupních transakcí. Algoritmus vstupního řízení se skládá z vzájemně kooperujících komponent - Predispatcher, Dispatcher a Monitorovací modulu a sady pravidel, které jsou v těchto komponentách využívány.



Obrázek 4.1: Schéma vstupního řízení

Cílem bylo navrhnout sadu pravidel, která by dokázala efektivně předcházet přetížení systému. Následující pravidla lze chápat jako jistý typ ovladače zahlcení nebo lépe množinu ovladačů zahlcení. Je však důležité zmínit, že tato pravidla ke svému vyhodnocení využívají informace o běhu systému Testbed. Na základě zkoumání systému a základních experimentů byla definována dvě pravidla, která jsou v rámci vstupního řízení aplikována. Snahou je minimalizovat nároky systému během provádění těchto pravidel.

Rozhodovací pravidla

Následující pravidla byla navrhnuta pro zachování maximální propustnosti systému, přičemž určitá skupina transakcí (dle vyhodnocení pravidel) není systémem zpracována. Při každém chodu systému je nastaveno, které pravidlo bude aplikováno. Pravidla vstupního řízení jsou aplikována při každém vstupu transakce do systému.

1. Jestliže platí $x > 2 * y$, kde *x* je počet zpožděných transakcí a *y* počet zpožděných transakcí u nepřetíženého systému, potom:

Aktuálně vstupující transakce do systému nebude zpracována. Tzn. bude označena jako ABORTED.

Při aplikaci pravidla je počet zpožděných transakcí nastaven na hodnotu 0.

2. Jestliže platí $x > 2 * y$, kde *x* je počet zpožděných transakcí a *y* počet zpožděných transakcí u nepřetíženého systému, a dále platí že:

Aktuálně vstupující transakce do systému bude mít kritický termín *d*:

$$d < \frac{deadline_{max} - deadline_{min}}{3} \quad ,$$

potom nebude transakce zpracována, a je označena jako ABORTED.

Při aplikaci pravidla je počet zpožděných transakcí nastaven na hodnotu 0.

Kdy jsou rozhodovací pravidla aplikována:

Schématicky lze znázornit navrhované vstupního řízení dle obrázku č. 4.1. Jde o spolupráci níže popsaných komponent systému a funguje následovně.

1. Při vstupu transakce je přijata modulem Predispatcher, tento modul zavolá funkci *LoadBalance*, která dle rozhodovacích pravidel (č. 1 nebo 2) rozhodne, zda bude transakce postoupena k dalšímu zpracování. Pokud je dle pravidel vyhodnoceno, že transakce nebude zpracována, je označena jako ABORTED. Pokud je dle pravidel vyhodnoceno, že bude transakce zpracována systémem, je poslána ke zpracování modulu *Dispatcher*.
2. Modul *Dispatcher* přiřadí prioritu dané transakci a předá ke zpracování modulem *Transaction Manager*.
3. Transaction Manager po zpracování jednotlivých částí, tzn. zpracování databázových operací, vyhodnotí, jestli daná transakce promeškala kritický termín nebo byla dokončena v kritickém termínu. Pokud transakce promeškala kritický termín, je *y*, tzn. počet všech transakcí, které nebyly dokončeny do kritického termínu navýšen o hodnotu 1.

V následujících kapitolách jsou popsány jednotlivé bloky zajišťující vstupní řízení. Bude vysvětlena implementace vstupního řízení založeného na zpětné vazbě systému a jak musí být pro tento typ vstupního řízení upraveny jednotlivé komponenty systému. Jednotlivé moduly jsou nezbytnou součástí systému vstupního řízení.

4.2 PREDISPATCHER

Hlavním úkolem modulu Predispatcher je zabránit přetížení systému. Dle definovaných pravidel rozhodovat, které transakce budou systémem zpracovány, případně které budou předčasně ukončeny.

Nastavení modulu:

- **Priorita:** N + 1, kde N je nejvyšší priorita používaná pro úlohy zpracovávající transakce. Tato reálná priorita procesu je nejvyšší oproti ostatním úlohám v rámci Testbedu. Se stejnou prioritou jsou spuštěni virtuální klienti, kteří zajišťují generování vstupních transakcí.

Vstupy modulu:

- **počet transakcí, které promeškaly kritický termín**
- **vstupní transakce**, posílány lokálními klienty
- **RT charakteristiky**, kritický termín a kritičnost
- **průměrný počet DB operací na transakci**

Výstupy modulu:

- **Označení vstupující transakce**
  - postoupena ke zpracování
  - nezpracována kvůli přetížení systému
- **Aktualizované informace k transakci v TRX_INFO**
  - doba zpracování modulem Dispatcher

Popis funkcí modulu:

ReadTrx()

- Načtení transakce ze vstupní fronty zpráv (FIFO)

SaveTrx()

- Uložení transakce do struktury TRX_INFO
- Uložení transakce do seznamu aktuálně zpracovávaných transakcí

LoadBalance()

- Vyhodnocení pravidla pro vstupní řízení a rozhodnutí, zda bude transakce zpracována. Odeslání transakce ke zpracování modulu Dispatcher. Pokud není transakce zpracována je označena jako tzv. ABORTED. Při vyhodnocování pravidel je programově zakázáno přepnutí kontextu úlohy.

4.3 DISPATCHER

Popis modulu:

Modul dle zvoleného algoritmu přidělování priorit přiděluje daným transakcím reálnou prioritu a spouští daný proces vykonávající transakci. V navrženém algoritmu vstupního řízení se zkoumá pouze varianta zpracování transakce v samostatném vlákně.

Nastavení modulu:

- **Priorita úlohy:** N + 1, kde N je nejvyšší priorita používaná pro úlohy zpracovávající transakce.
- **Pro zpracování transakce v modulu Transaction Manager** je zvolena možnost „**trx per process“** neboli pro každou transakci je vytvořeno samostatné vlákno pro zpracování jednotlivých databázových operací.
- **Algoritmus přidělování priorit:**
  - V testech byl využit algoritmus DM, který přiřazuje prioritu podle kritického termínu transakce, více v 2.4.2

Vstupy modulu:

- Vstupní transakce generované virtuálními klienty
- Informace z monitorovacího modulu

Výstupy modulu:

- Vytvořené procesy (v prostředí RTX jde o vlákna), které vykonávají jednotlivé transakce
- Aktualizované informace k transakci, doba zpracování modulem Dispatcher
- Transakce, které nemohly být restartovány dostanou zvláštní příznak

Popis funkcí daného modulu:

SetPriority(trx)

- Nastaví prioritu dle zvoleného algoritmu
- Uchovává informace o aktuálně přidělených prioritách v daném systému (uchovává daný rozsah)

CreateTask(trx)

- vytvoř vlákno ve kterém jsou jednotlivé databázové operace zpracovány

4.4 MONITOROVACÍ SUBSYSTÉM

Jedním z cílů této práce je analýza, návrh a zpracovaní monitorovacího subsystému, který umožní zjišťovat aktuální informace o systému. Modul zahrnuje monitorování aktuálního stavu databázového systému a zpracovávaní transakcí. Získané informace jsou využívány moduly *Predispatcher* a *Transaction Manager*. Základním kritériem při návrhu monitorovacího modulu bylo minimalizovat nutné systémové zdroje pro jeho provoz a tím minimalizovat vliv na chod modulu a celého systému Testbed.

Vstupy modulu:

- **vstupující transakce do systému a ukončená transakce**
- **zpožděná transakce**

Výstupy modulu:

- Počet trx, které promeškaly svůj kritický termín
- Počet aktuálně zpracovávaných transakcí

Popis funkcí daného modulu:

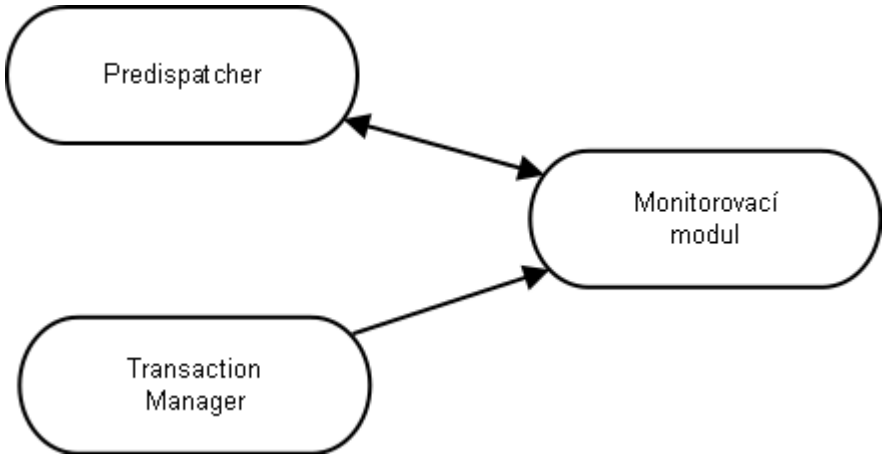
GetMissedDeadlineTrxCount()

- Zjišťuje počet transakcí, které zmeškaly svůj kritický termín

GetCurrentProcessedTrxCount()

- Zjistí počet aktuálně zpracovávaných transakcí

Z pohledu výměny informací mezi moduly, bychom mohli schématicky popsat vstupy a výstupy mezi moduly pomocí obrázku 4.2. Informace o stavu systému ukládají moduly Predispatcher a Transaction manager.



Obrázek 4.2: Monitorovací modul a spolupráce s ostatními moduly

5. TESTOVÁNÍ VSTUPNÍHO ŘÍZENÍ A VALIDACE SYSTÉMU

Proto, abychom mohli zjistit parametry daného systému, je nutné přistupovat systematicky k testování. Pomineme-li oblast funkcionálního testování, je pro nás důležité **zátěžové testování** (anglický ekvivalent **Load** či **Stress Testing**), což je proces tvorby požadavků na systém a měření rychlosti odezvy, v našem případě měření hodnotících kritérií pro zpracovávané transakce.

Systém Testbed byl při své inicializaci spuštěn v tzv. **testovacím módu**, kdy byly doladěny parametry jednotlivých pravidel a nastavení. Specifika testovacího módu:

1. Velmi obecné nastavení pravidel a systémového nastavení
2. Ukládání a vyhodnocení transakcí, které nejsou systémem zpracovány
3. Informace nutné k další analýze, spuštění systému v „debugging režimu“ a zaznamenávání maximálních informací o chování systému.

Informace, které sledujeme a s kterými pracujeme během jednotlivých testů:

1. Různé parametry vstupních transakcí a nastavení jednotlivých modulů systému
2. Zjištění vlivu monitorovacího modul na chod systému a režie nutná k zjišťování daných informací, ať už vypisováním na obrazovku či do logovacího souboru
3. Zjištění stavu systému, kde je již znatelný vliv degradace na propustnost systému
4. Testování vstupního řízení s různými druhy vstupních transakcí
5. Otestování algoritmu na různé typy vstupních transakcí s různým počtem databázových operací

Zachování stejných podmínek prostředí pro všechny prováděné testy:

1. Před každým testem byl operační systém MS Windows XP restartován a uveden do stejného výchozího stavu, s touto operací je spojena také akce restartování platformy RTX
2. Všechny testy byly provedeny 5 krát a jejich výsledky vždy zprůměrovány

5.1 PROSTŘEDKY PRO TESTOVÁNÍ

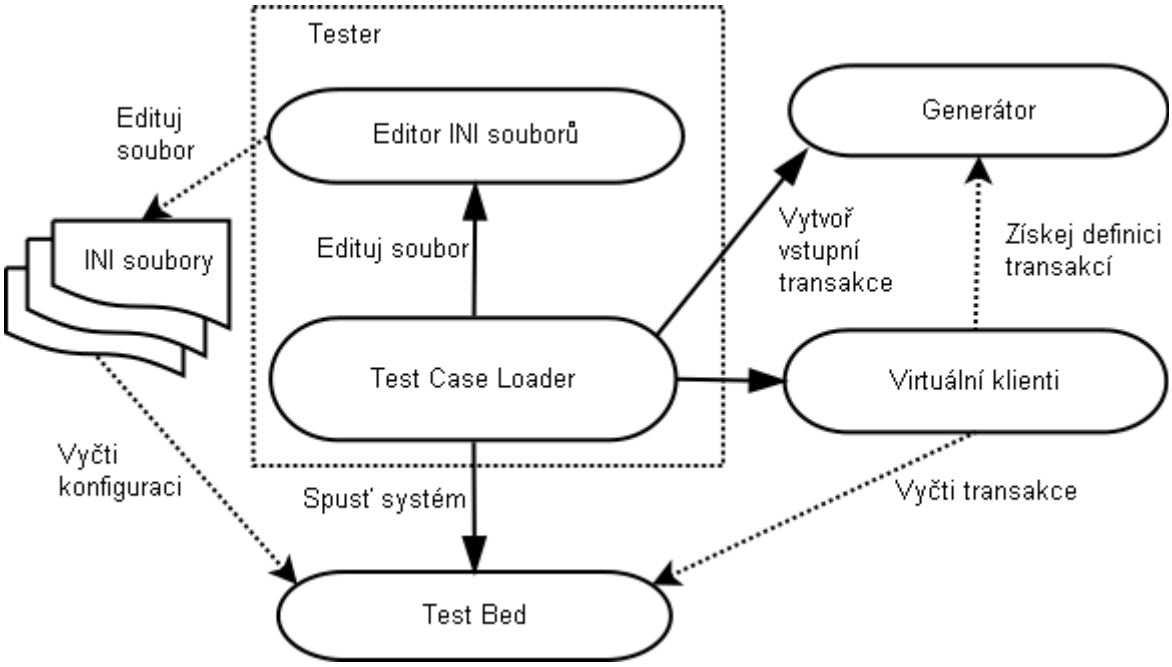
Testování systému Testbed bylo usnadněno díky používání pomocného nástroje s názvem **Tester**[30]**,** který byl vyvinut Tomášem Adámkem[31]. Tato utilita umožnila automatické provádění jednotlivých testovacích případů. Nástroj Tester spravuje jednotlivé testovací plány, tzn. umožní editovat vstupy a opětovně spouštět systém Testbed. Všechny tyto informace pro přehlednost ukládá na jednom místě, a tím umožňuje opětovné spuštění testovacího případu.

[30] Aplikace byla vyvinuta pro univerzální testování, byla ale přizpůsobena testování systému Testbed. Aplikace byla vyvinuta v jazyce Java, http://www.java.com.

[31] Student Katedry informatiky, Fakulty elektrotechniky a informatiky, Vysoké školy báňské – Technické univerzity Ostrava.

Nástroj Tester se skládá ze dvou komponent:

- **Editor INI souborů**, který umožňuje vytváření a editaci inicializačních souborů.
- **Test Case Loader**, který spouští systém Testbed s připravenými inicializačními soubory a případně ukládá výsledky chodu systému do databáze.



Obrázek 5.1: Nástroj Tester a spolupráce s okolím

Vlastnosti nástroje Tester:

- Nastavení cesty k systému Testbed a inicializačním souborům, Testbed je spouštěn dávkovým souborem typu *.bat
- Možnost vytvářet testovací případy a automaticky je spouštět, editace konfiguračních souborů z jednoho místa
- Vytvářet nové testovací případy kopírováním již existujícího testu
- Zautomatizovat spouštění jednotlivých testů přímo z prostředí Tester
- Opakované spouštění vytvořených testovacích případů
- Výsledky uložit ve specifikovaném adresáři, případně výsledky uložit do databázové systému pro další analýzu[32]

[32] Zjišťování skrytých závislostí a analýza výstupních dat ze systému Testbed je doporučována k navazujícímu výzkumu. Ukládání výsledku nebylo prozatím do systému Tester implementováno.

Postup práce s nástrojem Tester:

1. Vytvoření nového testovacího případu
2. Vytvoření, editace a uložení inicializačních souborů
3. Kontrola správné syntaxe vstupních souborů
4. Spuštění systému Testbed s konfiguračními soubory a soubory definujícími vstupní transakce
5. Ukončení systému Testbed a testu
6. Uložení výstupního souborů trx_info.csv do speciálního adresáře
7. Vytvoření dalšího testovacího případu či znovu spuštění testu

5.1.1 Popis hardware

Testy popsané v následujících kapitolách byly prováděny na této hardware konfiguraci:

- Notebook DELL Latitude D830, vybaven procesorem Core 2 Duo T7100 s frekvencí 1.8Ghz a 1 GB RAM paměti

5.1.2 Nastavení platformy RTX

Provádění jednotlivých procesů bylo založeno na preemptivním přepínáním procesů podle priorit jednotlivých úloh. Před každým experimentem byl systém restartován, aby bylo zajištěno vždy stejné výchozí prostředí pro jednotlivé testy. Testy byly provedeny v operačním systému RTX, Real-Time eXtension 8.1 s následujícím nastavením:

- **Platforma reálného času:** RTX verze 8.1, build 6224, Multiprocessor configuration, dedicated processor
- **Operační systém**: Windows XP SP3
- **Vývojové prostředí:** MS Visual Studio 2005 Professional Edition

Hal timer period	100 [μs]
Time quantum	0
Free stack on Terminate Thread calls	Vypnuto
Delay full cache flush while RTSS applications are running	Zapnuto
RTSS process slots	10
Memory → Local Memory Pool	104857600 [bytes]
Mapped Memory	64000000 [bytes]

Tabulka 5.1: Nastavení prostředí RTX

5.2 NASTAVENÍ SYSTÉMU TESTBED

Tabulka níže popisuje základní nastavení pro všechny testovací případy. Pokud byl během testu nějaký parametr systému Testbed změněn, je to vždy explicitně uvedeno před testem. Systém Testbed nabízí více parametrů nastavení, níže jsou uvedeny pouze nejdůležitější parametry související s testovacími případy v následujících kapitolách.

Konfigurace komponent systému	Hodnota	Popis
PREDISPATCHER		
PREDISPATCHER_PRIORITY	127	Priorita procesu predispatcher
PREDISPATCHER_AC_TYPE	1 nebo 2	Typ pravidla vstupního řízení
TRASHING_PROTECTION	Zapnuto	Zapnuto/Vypnuto zabránění zahlcení systému
PREDISPATCHER_TYPE_MSG		Transakce jsou posílány přes fronty zpráv
DISPATCHER		
DISPATCHER_PRIORITY	127	Priorita úlohy dispatcher
DISPATCHER_METHOD	DM	Deadline Monotonic. Typ metody přidělování priorit.
DISPATCHER_TRX_PRIORITY_TYPE	DEADLINE	Využívá k určení priority úlohy pouze kritický termín
TRX_REAL_PRIORITY_LOW	1	Nejnižší reálná priorita RTX používaná pro transakce
TRX_REAL_PRIORITY_HIGH	126	Nejvyšší reálná priorita RTX používaná pro transakce
TRX_PROCESSING_TYPE_PROCESS_TRX		Každá transakce je zpracována jako samostatná úloha/vlákno v prostředí RTX
OSTATNÍ KOMPONENTY SYSTÉMU		
MEMORY_INDEX_METHOD	0	Nastavuje typ metody použité pro indexování dat uložených v paměti, indexování není nastaveno
HDD_INDEX_METHOD	0	Nastavuje typ metody použité pro indexování dat uložených na disku, indexování není nastaveno
CC_METHOD	1	2LP-HP

Tabulka 5.2: Nastavení systému Testbed

5.2.1 Vstupní transakce

Vytvořený experimentální systém Testbed poskytuje variabilitu při nastavení vstupních transakcí, což se během provádění jednotlivých experimentů ukázalo jako velmi výhodné. Transakce je možné generovat náhodně v předem definovaných časových intervalech nebo periodicky. Následující text definuje všechny parametry transakcí, které byly použity pro prováděné experimenty.

Interval pro vstup transakcí do systému:

Všechny vstupní transakce jsou definovány jako náhodné, přicházejí do systému v určitém časovém rozsahu, daném parametry L a H. V rámci tohoto časového rozsahu (časové hodnoty jsou v milisekundách) jsou transakce generovány na základě rovnoměrného rozdělení. Tím bylo možné docílit toho, že známe počet transakcí za určitý čas a transakce nepřicházejí do systému najednou, ale postupně. Například pro základní interval při testování přetížení systému v kapitole č. 5.5 je spodní hranice časového rozsahu L = 0,5 ms a horní hranice H = 25 ms. Průměrně vytvoří takto definovaný generátor 80 transakcí za sekundu, tzn. jednu transakci za 12,5 ms.

Nastavení pro generování vstupních transakcí:

Při testování přetížení systému byl využit jeden nebo více klientů, kteří generovali transakce do systému. Obecně lze generování transakcí do systému nastavit následovně:

- jeden virtuální klient s x transakcemi, kde $x \geq 1$
- x virtuálních klientů s y transakcemi, kde $y \geq x \geq 1$

Počet vstupních transakcí:

Při prováděných experimentech bylo použito 1500 vstupních transakcí. Po ukončení běhu systému byl obsah struktury TRX_INFO, obsahující informace o zpracovaných transakcích, uložen do *.csv souboru pro další vyhodnocení.

$$počet\ vstupních\ transakcí = [1500]$$

Databázové operace

Část definice transakce tvoří databázové operace. Vzhledem k charakteru experimentů byly použity operace základní databázové operace INSERT, UPDATE, SELECT, DELETE.

$$Počet\ databázových\ operací\ v\ transakci \sim [5,10]$$

Databázové schéma

Počet tabulek je v jednotlivých testech nastaven stejně, databáze se skládá z 20-ti tabulek, přičemž každá z nich má 5 atributů definovaného datového typu. Databáze tedy byla v rámci experimentů chápána jako oblast s definovaným počtem prvků. Transakce nad touto databází prováděly základní databázové operace pro čtení a zápis hodnot.

5.2.2 Real-Time charakteristiky

Pro všechny testovací případy byly real-time charakteristiky transakce definovány následujícím způsobem.

Kritický termín (deadline)

Nastavení kritického termínu pro provádění transakcí má výrazný vliv na výsledky experimentů. V rámci definice transakce se kritický termín chápe jako relativní čas od počátku přijetí transakce, do kterého musí být daná transakce zpracována, aby byla považována za úspěšně zpracovanou. Parametr je nastavován v určitém rozsahu a je zapsaný v definici transakce hodnotou za znakem D. Hodnota kritického termínu je uváděna v milisekundách. Experimentálně byl zjištěn průměrný čas, potřebný pro provedení jedné transakce obsahující 5 databázových operací, v případě nezatíženého systému. Pro krátké transakce, které byly definovány pro většinu experimentů, byla tato doba rovna 1,5 ms. Rozsah pro náhodné nastavení kritického termínu je daný násobky hodnoty této hodnoty po zaokrouhlení. Jako spodní mezní hodnota byla zvolena 1, pro horní mez je rovna násobku hodnoty 10.

$$Kritický\ termín \sim [1 * průměrná\ doba\ zpracování\ trx, 10 * průměrná\ doba\ zpracování\ trx]$$

$$Kritický\ termín \sim [2,20]$$

Kritičnost (criticality)

Druhý parametr RT charakteristiky transakce udává její důležitost vůči ostatním transakcím. V rámci všech experimentů byl tento parametr pro transakce náhodně generován na základě rovnoměrného rozdělení v rozsahu [1, 10], přičemž hodnota 1 znamená nejvyšší důležitost a hodnota 10 nejnižší.

$$Kritičnost\ C \sim [1,10]\ (1 - nejvyšší, 10 - nejnižší)$$

Znalost těchto parametrů je z pohledu posouzení výsledků experimentů velmi důležitá.

5.3 HODNOTÍCÍ METRIKY

Základním kritériem pro posouzení kvality zpracování real-time databázových transakcí je především podíl splněných transakcí, tzn. poměr počtu zpracovaných transakcí před uplynutím kritického termínu vzhledem k celkovému počtu přijatých transakcí. Mimo toto hodnotící kritérium jsou sledovány a ve výstupních grafech prezentovány i další hodnotící metriky - zajímavé pro posouzení a porovnání v jednotlivých testovacích případech.

1. Úspěšnost transakcí

Podíl splněných transakcí je základním použitým ukazatelem, který slouží jako hodnotící kritérium ve většině experimentů.

$$Úspěšnost\ transakcí = \frac{\sum k}{\sum m} * 100\ [\%]\ \text{, kde}$$

k je transakce, která splnila svůj kritický termín; *m* je transakce, která byla zpracovaná systémem

2. Počet zpožděných transakcí

Za zpožděnou transakci považujeme takovou transakci, jejíž provedení trvalo déle než definovaný kritický termín.

$$Počet\ zpožděných\ transakcí = \sum i\ \text{, kde}$$

i je transakce, která byla zpracována systéme a nebyl dodržen kritický termín;

3. Počet nezpracovaných transakcí

Jedná se o počet transakcí, které nebyly díky vstupnímu řízení vpuštěny do systému Testbed, tzn. nebyly zpracovány.

$$Počet\ nezpracovaných\ transakcí = \sum j\ \text{, kde}$$

j je transakce, která nebyla zpracována

4. Průměrná doba zpracovaní transakce

Jedná se o průměrnou délku zpracování transakce. Průměrná délka zpracování transakce se mění s počtem transakcí, databázových operací a dalších vstupních parametrů.

$$Průměrna\ doba\ zpracování\ transakce = \frac{\sum t}{\sum m}\ \text{, kde}$$

t je doba zpracování transakce; *m* je transakce, která byla zpracovaná systémem

Všechny uvedené parametry jsou prezentovány ve výstupních grafech v závislosti na počtu příchozích transakcí nebo dalších vstupních kritériích. Struktura vstupních transakcí je pro každý experimentální případ explicitně popsána.

5.4 TESTOVACÍ PŘÍPADY

Testování je zaměřeno na zkoumání zpracování náhodně přicházejících transakcí. Systém Testbed je připraven také na zpracování periodických transakcí, jejich plánování se ale děje pomocí jiných protokolů přidělování priorit a většinou se jedná o hard real-time transakce, u kterých se apriori nepředpokládá promeškání kritického termínu. Z tohoto pohledu je výzkum a testování směřován především na aperiodické, tzn. náhodně generované transakce.

Z výzkumů předchozí verze platformy Testbed vyplývá, které protokoly souběžného řízení jsou vhodné pro různé typy vstupních transakcí. Výsledky testů jednotlivých statických algoritmů přiřazování priorit i s grafy a komentáři lze nalézt v (20). Na základě dosavadního vývoje a možností testování experimentálního systému vznikl požadavek na efektivnější práci se systémem při zatížení systému a různé intenzitě vstupů. Je proto důležité umět rozpoznat, kdy je systém přetížený a efektivně takovému stavu systému předcházet pomocí vhodně navrženého algoritmu vstupního řízení.

5.5 ZJIŠTĚNÍ PŘETÍŽENÍ SYSTÉMU TESTBED

Smyslem následujících testů bylo zjištění parametrů systému, za kterých dochází k přetížení, při variabilním vstupním intervalu vstupních transakcí, tzn. různém počtu vstupních transakcí a variabilním počtu klientů generujících transakce. V testech 1A, 1B a 1C není využito vstupní řízení, tzn. všechny příchozí transakce jsou systémem zpracovány.

Fixní parametry:

- priorita klienta = 127; počet transakcí = 10; počet opakování odesílání = 150
- počet databázových operací = 5
- počet vstupních transakcí = 1500; vyhodnocuje se prvních 500 příchozích transakcí

Variabilní parametry:

- intervaly, ve kterých jsou transakce náhodně posílány do systému: <0,5;25>; <0,5;15>; <0,5;10>; <0,5;7,5>; <0,5;5>; <0,5;2,5>; <0,5;1,5> <0,5,1>; <0,25-0,5>, hodnoty jsou v milisekundách
- variabilní počet databázových operací pro test 1C ~ [5,10]
- při generování transakcí pro test 1A byl použit jeden generátor transakcí, klient. Test 1B byl proveden pomocí tří klientů generujících nezávisle transakce do systému ve stejných časových intervalech

TEST 1A

V následujícím testu byl zjišťován podíl úspěšně provedených transakcí, tzn. do kritického termínu, při měnícím se intervalu generování vstupních transakcí a zvyšování počtu vstupních transakcí v časovém intervalu. Transakce byly náhodně generovány do systému v počtu od 80 po 4000 transakcí za sekundu. Pro získání náhodné hodnoty v rámci použitého intervalu byl využit generátor náhodných čísel, který zajišťuje rovnoměrné rozdělení získaných hodnot[33]. Testy byly prováděny postupně od nejdelšího intervalu po nejkratší. První interval <0,5;25> byl zvolen na základě výsledku experimentu, který ukázal, že systém dokáže úspěšně zpracovat 100% vstupních transakcí při daném nastavení. Základní experimenty také ukázaly, že průměrná délka zpracování transakce je cca 1,5 ms.

Fixní parametry:

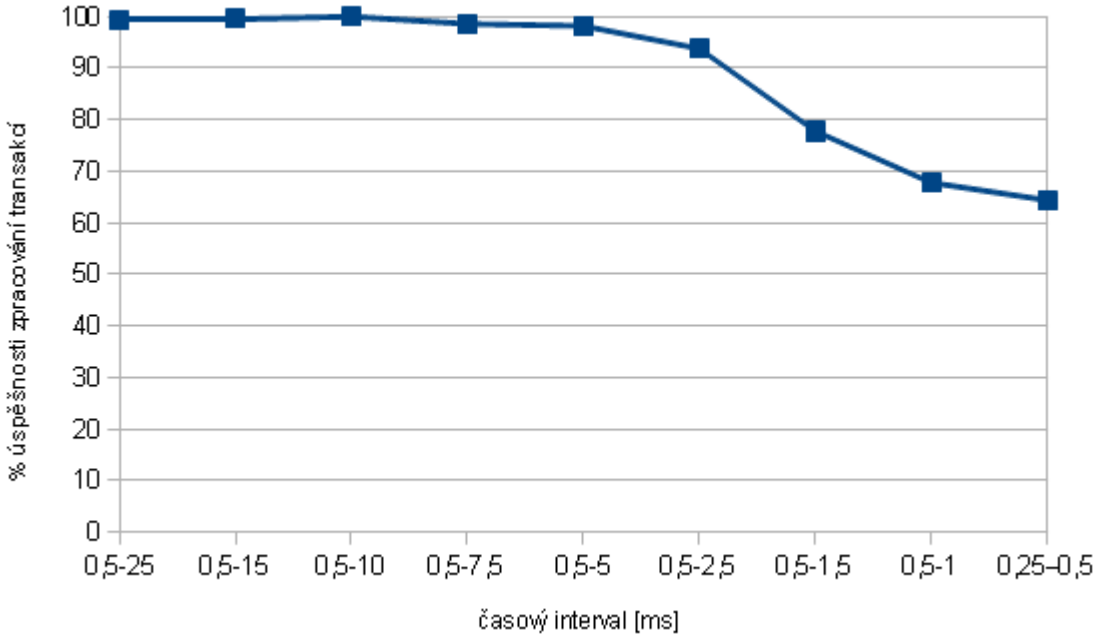
- počet vstupních transakcí = 1500, přičemž je vyhodnoceno prvních 500 transakcí, které vstoupily do systému
- počet databázových operací v transakci = 5
- počet klientů generujících vstupní transakce = 1

Variabilní parametr:

- intervaly pro generování transakcí do systému: <0,5;25>; <0,5;15>; <0,5;10>; <0,5;7,5>; <0,5;5>; <0,5;2,5>; <0,5;1,5>; <0,5;1>; <0,25;0,5>, hodnoty jsou v ms

Sledované veličina, hodnotící metrika:

- **Úspěšnost transakcí** [%], tzn. procento úspěšně zpracovaných transakcí do kritického termínu



Graf 5.1: Úspěšnost zpracování transakcí

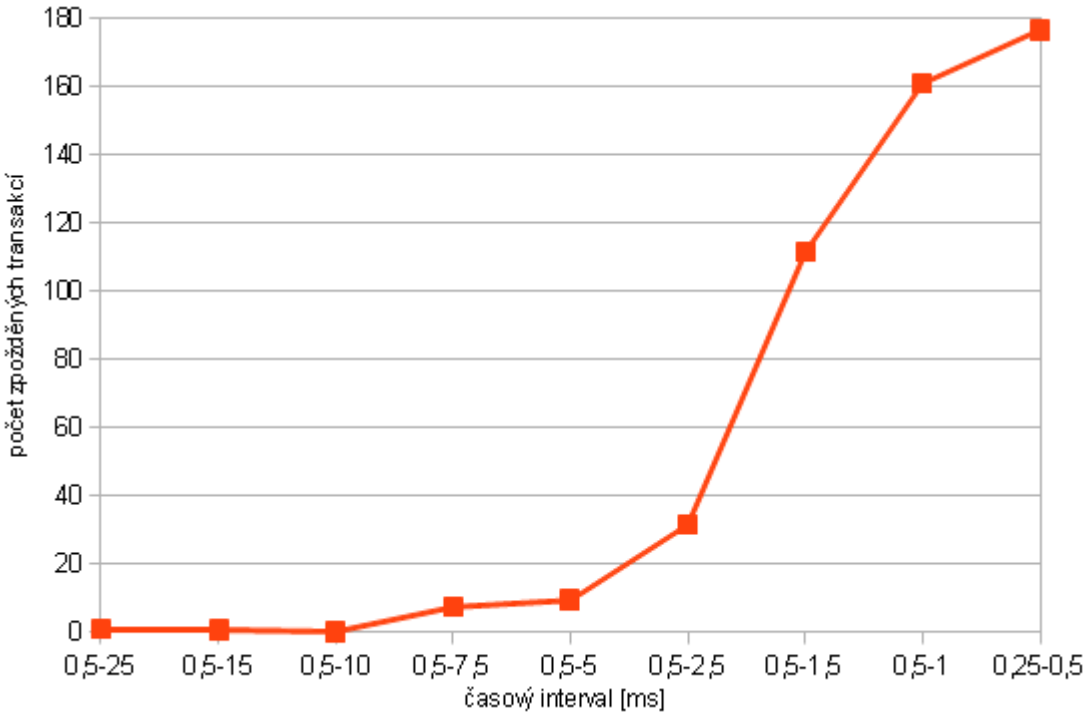
[33] Viz kapitola 3.1.

Komentář výsledků:

Z grafu č. 5.1 vyplývá, že systém je jednoznačně nevytížený – je schopen zpracovat všechny vstupní transakce na základně kritického termínu, pokud přijímá méně transakcí než 130 za sekundu. Při generování 130–400 transakcí za sekundu lze vidět v řádu jednotek transakce, které promeškaly kritický termín. Při náhodném generovaní transakcí v intervalu 0,5-5 ms, tzn. 400 transakcí za sekundu a více lze sledovat významnou změnu vytížení systému. Při dalším zvyšování počtu transakcí a zpracování 600 a více transakcí za sekundu, je zřejmé, že procento úspěšně zpracovaných transakcí klesá lineárně až k 65 procentům. Při generování v intervalech kratších než 0,25 ms narážíme na limity systému, proto nejsou kratší intervaly v testu zahrnuty.

Sledovaná veličina, hodnotící metrika:

- **Počet zpožděných transakcí,** tzn. počet transakcí, které byly zpracovány po kritickém termínu.



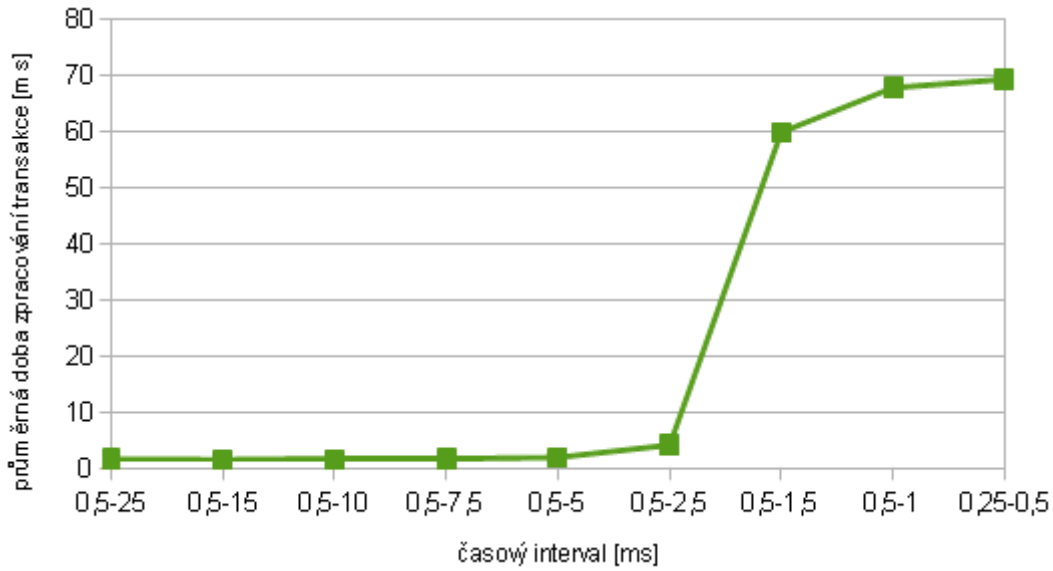
Graf 5.2: Počet zpožděných transakcí

Komentář výsledků:

Na grafu č. 5.2 lze vidět, že počet transakcí, které překročily kritický termín se zvyšuje při generování 250 a více transakcí za sekundu. Při náhodném generovaní transakcí v intervalu 0,5-5 ms, tzn. 400 transakcí za sekundu a více lze sledovat nárůst počtu zpožděných transakcí, což indikuje přetížení systému. V mezním intervalu se počet zpožděných transakcí přibližuje k hodnotě 180 transakcí, tzn. 36% všech vyhodnocovaných transakcí, přičemž se vyhodnocuje prvních 500 vstupních transakcí do systému.

Sledovaná veličina, hodnotící metrika:

- **Průměrná doba zpracování transakce**



Graf 5.3: Průměrná doba zpracování transakce

Komentář výsledků:

Z grafu č. 5.3 vyplývá, že průměrná doba zpracování transakce nezatíženého systému osciluje kolem hodnoty 1,5 ms. Při přetížení systému průměrná doba zpracování transakce dosahuje hodnoty až 70 ms. To je v porovnání s průměrnou dobou zpracování transakce nezatíženého systému více než 46 krát delší čas. Tento ukazatel jednoznačně zachycuje okamžik přetížení systému, kdy se prodlužuje průměrná doba zpracování transakce a dochází k degradaci systému.

TEST 1B

V následujícím testu byl zjišťován podíl úspěšně provedených transakcí při měnícím se intervalu generování vstupních transakcí a zvyšování počtu vstupních transakcí v časovém intervalu. Transakce byly náhodně generovány do systému v počtu od 80 po 4000 transakcí za sekundu. V následujícím testu byli použiti 3 klienti pro generování stejného počtu transakcí, jako v předchozím testu 1A. Do systému tedy transakce přicházely s trojnásobnou intenzitou, tzn. ve stejném intervalu vstoupily do systému 3 transakce v testu 1B, přičemž v testu 1A pouze 1 transakce. To je také důvod, proč je interval hodnot odlišný a mezní hodnota intervalu pro generování transakcí u testu 1B dosahuje <0,5;2>.

Fixní parametry:

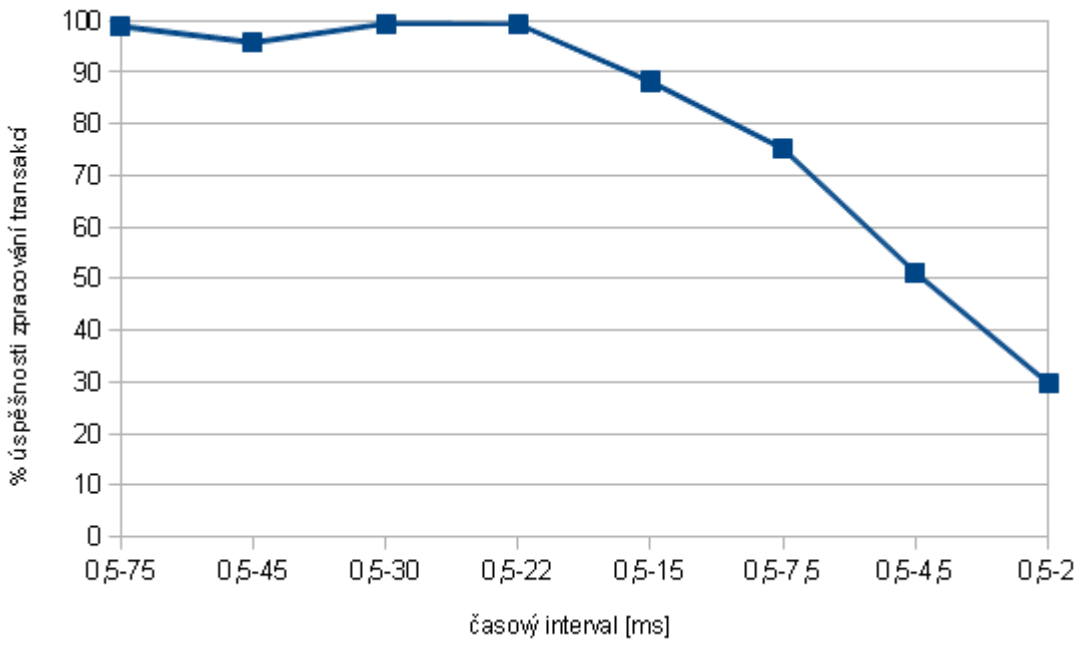
- počet vstupních transakcí = 1500, přičemž je vyhodnoceno prvních 500 transakcí, které vstoupily do systému
- počet DB operací transakce = 5
- počet klientů generujících vstupní transakce = 3 s prioritou = 127

Variabilní parametr:

Testy byly prováděny postupně od nejdelšího intervalu po nejkratší: <0,5;75>; <0,5;45>; <0,5,30>; <0,5;22>; <0,5;15>; <0,5;15>; <0,5;4,5>; <0,5;2>. První interval <0,5;75> byl zvolen na základě výsledku experimentu, který ukázal, že systém dokáže úspěšně zpracovat 100% vstupních transakcí při daném nastavení. Základní experimenty také ukázaly, že průměrná délka zpracování transakce je cca 1,5 ms.

Sledovaná veličina, hodnotící metrika:

- **Úspěšnost transakcí** [%], tzn. procento úspěšně zpracovaných transakcí do kritického termínu.



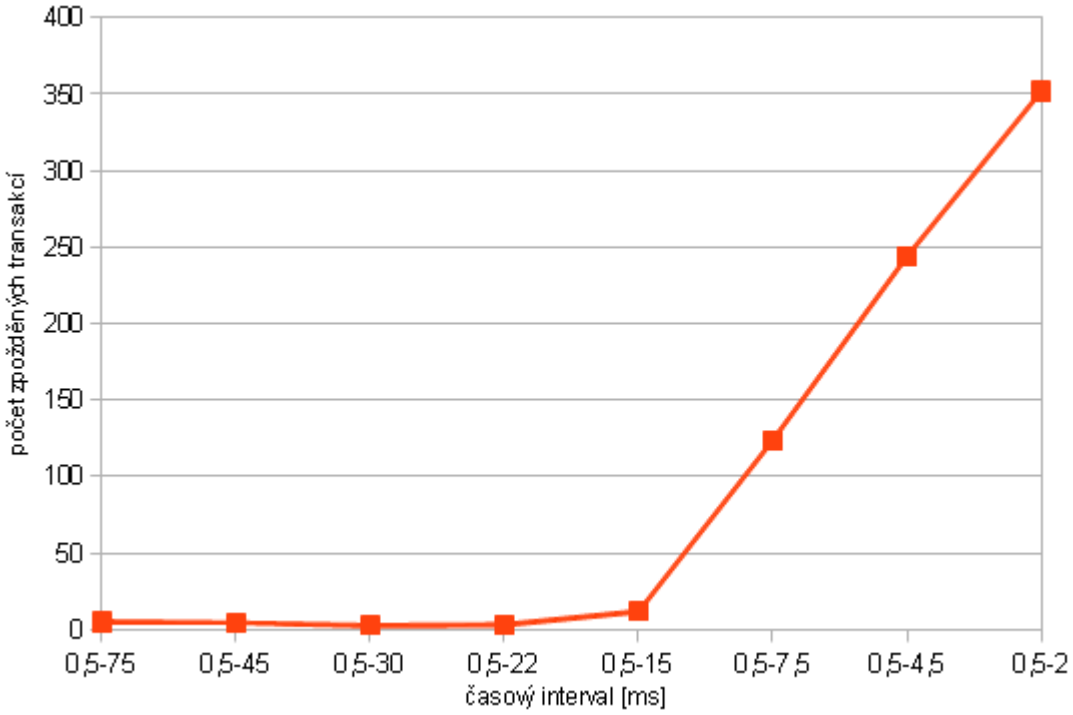
Graf 5.4: Úspěšnost zpracování transakcí

Komentář výsledků:

V testu 1B byl zpracován stejný počet, stejně definovaných transakcí, jako v testu 1A. Jediným rozdílem je, že transakce byly posílány do systému paralelně pomocí tří klientů, proto jsou transakce posílány v delších intervalech. Zajímavým výsledkem testu je, že dochází k větší zátěži a následné degradaci systému při generování maximálního počtu transakcí, tedy 4000 transakcí za sekundu. V mezním intervalu 0,5–2 ms klesne procento úspěšně zpracovaných transakcí v rámci kritického termínu až k hodnotě 30% v porovnání s 65% v testu 1A.

Sledovaná veličina, hodnotící metrika:

- **Počet zpožděných transakcí**, tzn. počet transakcí, které byly zpracovány po kritickém termínu.



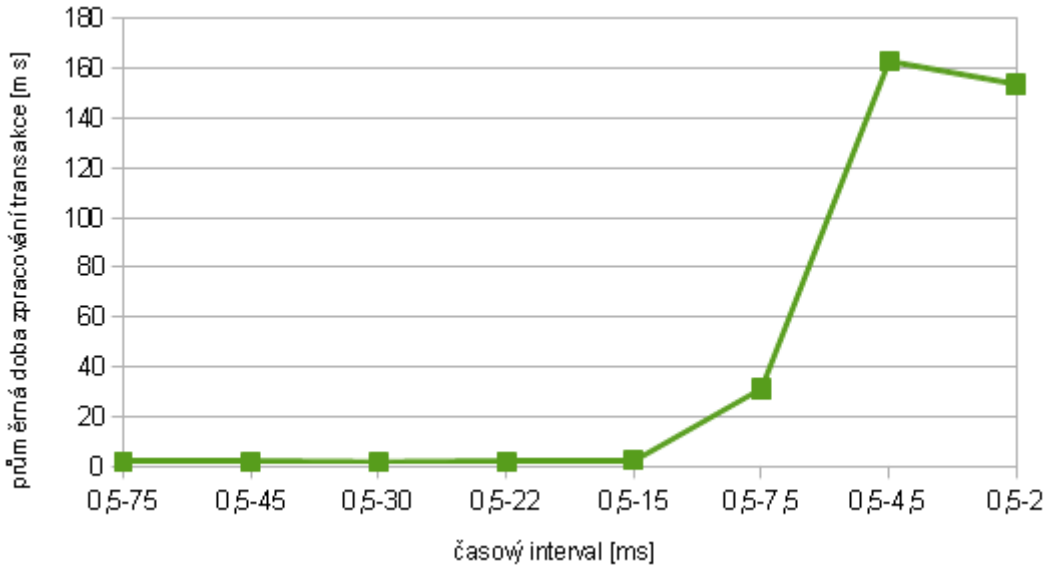
Graf 5.5: Počet zpožděných transakcí

Komentář výsledků:

Z výsledku testu 1B a hodnot z grafu 5.5, lze pozorovat, že počet zpožděných transakcí v mezním intervalu <0,5;2> se blíží k hodnotě 350 transakcí. Tento počet odpovídá 70% všech vyhodnocovaných transakcí. Lze tedy rozpoznat znatelné přetížení systému v této mezní hodnotě.

Sledované veličina, hodnotící metrika:

- **Průměrná doba zpracování transakce**



Graf 5.6: Průměrná doba zpracování transakce

Komentář výsledků:

Z grafu 5.6 vyplývá, že průměrné doby zpracování transakcí při přetížení systému v nejkratších intervalech dosahují hodnot nad 150 ms. Tato hodnota je v porovnání s průměrnou dobou zpracování transakcí, kdy není systém přetížen, přibližně 100 násobně vyšší. Lze konstatovat, že systém je přetížen a dochází k degradaci zpracovávaných transakcí.

TEST 1C

Následující test má za cíl ukázat, jak je průměrná doba zpracování transakce ovlivněna počtem databázových operací v transakci.

Fixní parametry:

- na základě výsledků testu 1A byly transakce generovány náhodně v intervalu <1;25> [ms]. Při tomto intervalu generování transakcí, stejné definici transakcí a nastavení systému Testbed, lze na základě předchozích testů považovat systém za nepřetížený
- počet klientů generujících transakce = 1
- celkový počet vstupních transakcí = 1500, přičemž se výsledky vyhodnocují pro prvních 500 příchozích transakcí

Variabilní parametr:

- počet transakcí 5, 10

Sledovaná veličina, hodnotící metrika:

- **Průměrná doba pro zpracování transakce**

Počet databázových operací	Průměrná délka zpracování transakce
5	1,5 [ms]
10	3,8 [ms]

Tabulka 5.3: Průměrná doba zpracování transakce při různém počtu operací

Komentář:

Výsledky testu 1C dokazují, že počet DB operací výrazně ovlivňuje dobu zpracování transakce. Průměrná délka zpracování transakce při 5 DB operacích = 1,5 ms, při 10 databázových operací se zvýší doba zpracování na 3,8 ms.

Vyhodnocení testů 1A, 1B, 1C:

Dle sledovaných metrik a na základě výsledků testu 1A, zobrazených grafy č. 5.1, 5.2, 5.3, lze konstatovat, že systém je nepřetížený, pokud zpracovává 130 a méně transakcí za sekundu. Při generování 130–400 transakcí za sekundu lze vidět, že systém není schopen zpracovat v kritickém termínu transakce v řádu jednotek. Při zpracování 400 a více transakcí za sekundu již znatelně přibývá transakcí, které jsou zpracovány po svém kritickém termín. Výsledky testu 1B jednoznačně potvrzují, že více klientů paralelně generujících transakce negativně ovlivňuje činnost systému. Potvrzují to především metriky úspěšnosti zpracovaných transakcí a průměrná doba zpracování transakcí v testech 1A a 1B. Na základě těchto výsledků byl pro další testy zvolen pouze jeden klient generující transakce. Výsledky testů 1A, 1B a 1C sloužily k nastavení parametrů systému pro testy v následující kapitole.

5.6 TESTOVÁNÍ ALGORITMU VSTUPNÍHO ŘÍZENÍ

Zjištění vlastností algoritmu vstupního řízení založeného na zpětné vazbě systému

TEST 2A

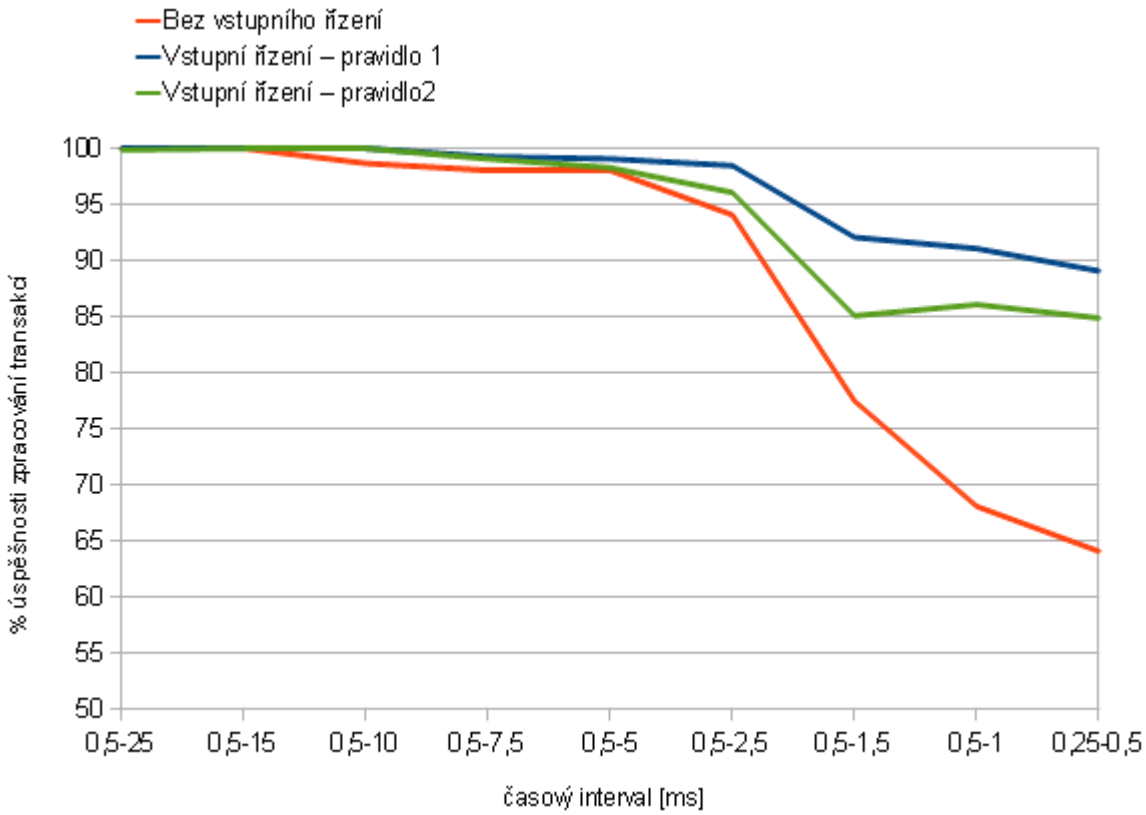
Smyslem tohoto testu je zjištění funkce vstupního řízení při přetížení systému, přičemž do systému budou vstupovat transakce se stejným počtem DB operací. Snahou je zjistit, zda díky algoritmu vstupního řízení dokáže systém Testbed odolat zvyšující se zátěži a předcházet přetížení. Vstupní řízení bylo testováno s pravidlem č. 1, tzn. „nevpusť do systému transakce s nízkým kritickým termínem“ a pravidlem č. 2, „nevpusť/nezpracuj jakoukoliv vstupní transakci“ a za předpokladu, že aktuální počet promeškaných transakcí je větší než 5. Tato konstanta byla vypočítána na základě testu 1A a předpokladu, že 2 transakce mohou být opožděny i při běžném chodu systému.

Vstupní parametry:

- počet databázových operací v transakci = 5
- počet aperiodických transakcí = 1500
- intervaly, v kterých jsou náhodně posílány transakce do systému: <0,5;25>; <0,5;15>; <0,5;10>; <0,5;7,5>; <0,5;5>; <0,5;2,5>; <0,5;1,5>; <0,5;1>; <0,25;0,5>, hodnoty v ms

Sledovaná veličina, hodnotící metrika:

- **Úspěšnost transakcí [%],** tzn. procento úspěšně zpracovaných transakcí do kritického termínu.



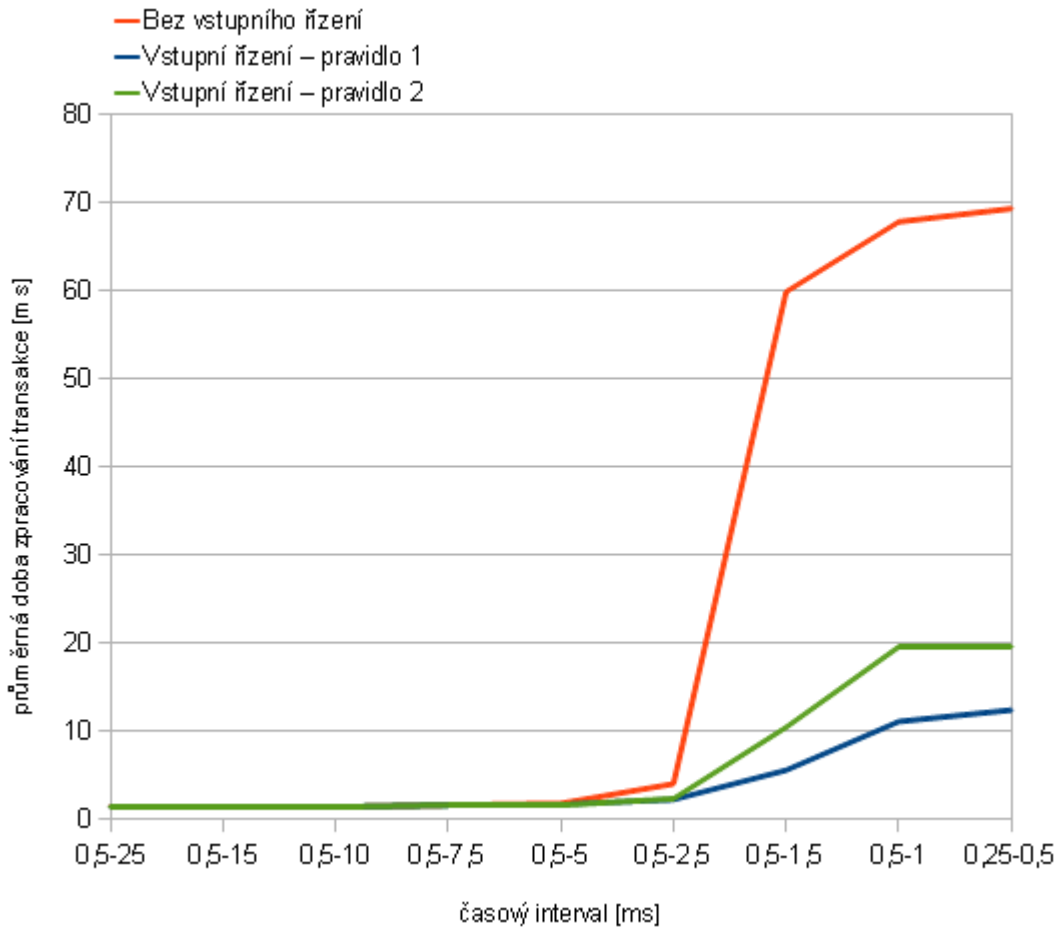
Graf 5.7: Úspěšnost zpracování transakcí

Komentář výsledků:

V grafu 5.7 lze pozorovat, že při aplikaci vstupního řízení neklesne procento úspěšnosti zpracovaných transakcí pod 85. Markantní je rozdíl při aplikaci pravidla č. 1, které zajišťuje vyšší propustnost systému, a to cca 87%. Dále lze pozorovat významný efekt, kdy vstupní řízení (pravidlo č.1 i 2) zajišťuje konstantní propustnost systému, tzn. nedochází k postupné degradaci zpracování vstupních transakcí.

Sledovaná veličina, hodnotící metrika:

- **průměrná doba zpracování transakce**



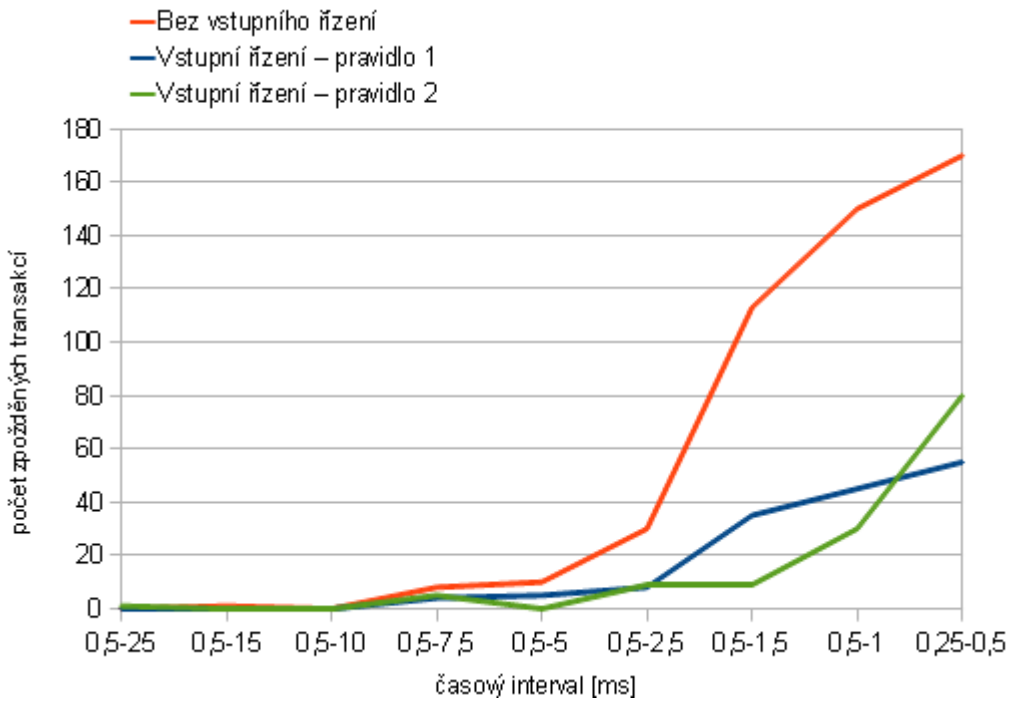
Graf 5.8: Průměrná doba zpracování transakce

Komentář výsledků:

Na základě grafu č. 5.8 lze konstatovat, že průměrná doba zpracování transakce se při využití vstupního řízení transakcí u zatíženého systému sníží a dosahuje maximálně doby cca 20 ms. Dále lze vidět, že lepších výsledků dosahuje pravidlo č.1, které do systému nevpustí transakce s vyšší prioritou.

Sledovaná veličina, hodnotící metrika:

- **počet zpožděných transakcí**



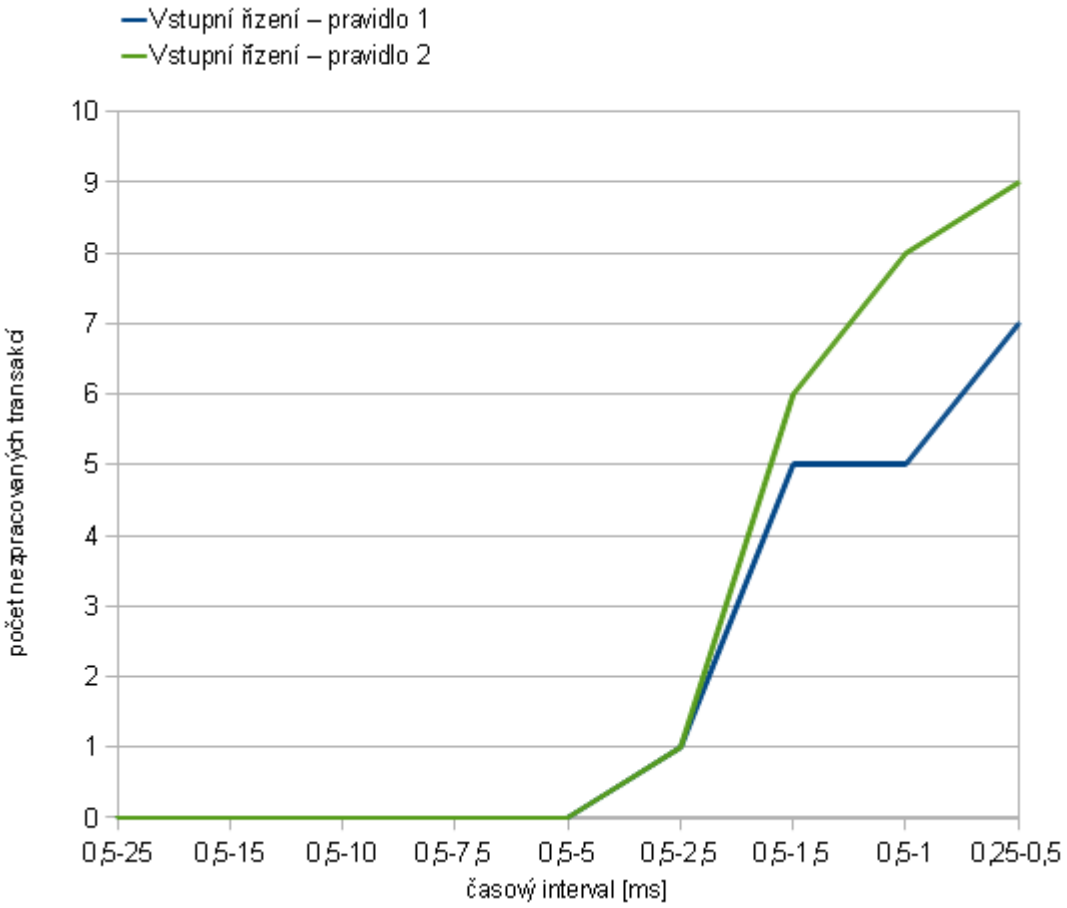
Graf 5.9: Počet zpožděných transakcí

Komentář výsledků:

Dle grafu 5.9 lze zjistit, že u systému bez vstupního řízení se může zpozdit až 36 % vstupních transakcí. Pravidla vstupního řízení dokáží počet zpožděných transakcí radikálně eliminovat. Pro nižší zátěž se zdá výhodnější využití pravidla č. 1. Obecně lze konstatovat, že vstupní řízení dokáže eliminovat počet zpožděných transakcí na 16 % a méně. Při aplikaci vstupního řízení klesl počet zpožděných transakcí minimálně o 56 %, což lze vyhodnotit jako překvapivý výsledek s ohledem na výsledek testu 2B, který dokazuje, že používání algoritmu vstupního řízení a monitorovacího modulu neovlivňuje průměrnou dobu zpracování transakce.

Sledovaná veličina, hodnotící metrika:

- **počet nezpracovaných transakcí**



Graf 5.10: Počet nezpracovaných transakcí

Komentář výsledků:

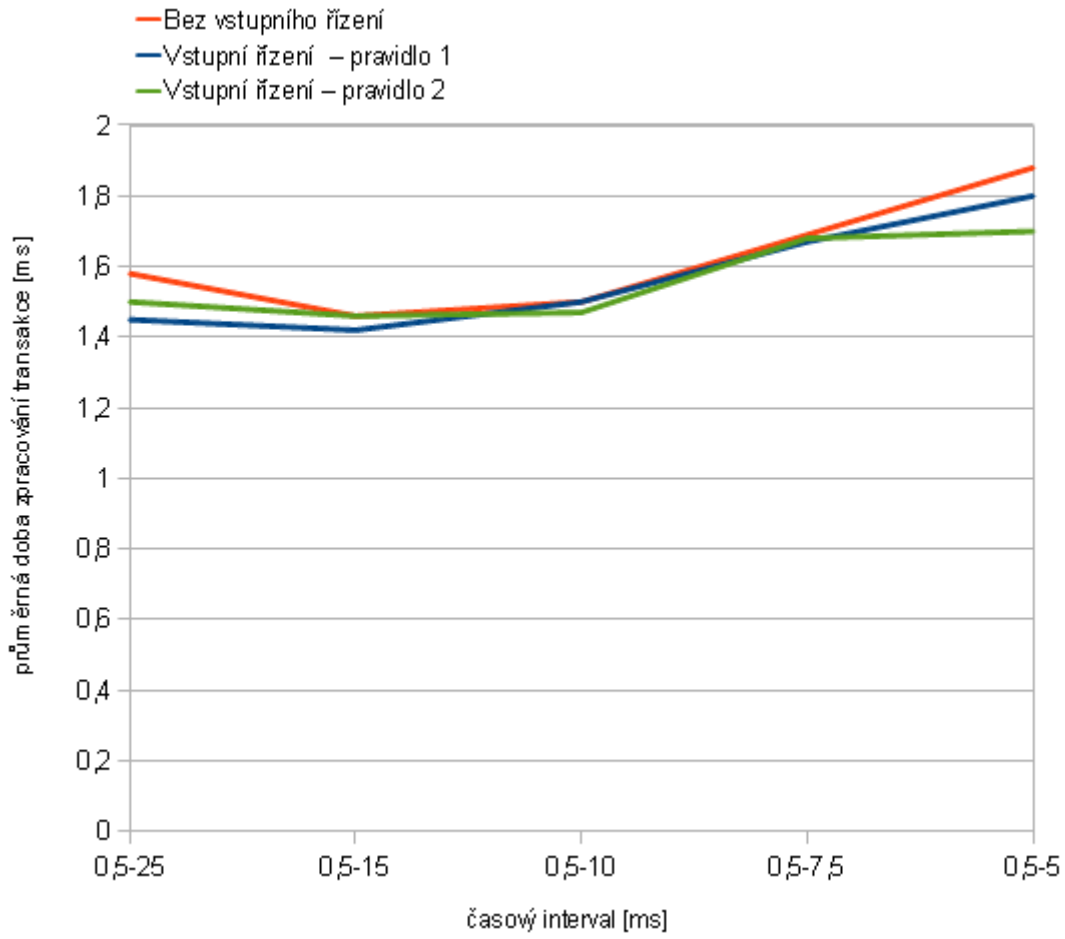
Pravidla algoritmu vstupního řízení rozhodují, která transakce nebude do systému vpuštěna a zpracována. V případě aplikace pravidla č. 1 není do systému vpuštěna transakce pouze s nízkým kritickým termínem, což znamená, že počet nevpuštěných transakcí do systému je nižší, ale výsledkem je vyšší počet úspěšně zpracovaných transakcí. V případě aplikace pravidla č. 2 není zpracována žádná transakce, pokud je aktuální počet promeškaných transakcí v systému vyšší než 5. Výsledný počet nezpracovaných transakcí je v případě aplikace pravidla č. 2 vyšší. Za velmi zajímavý jev lze považovat, že při relativně malém počtu nezpracovaných transakcí u obou pravidel, byly výsledky úspěšnosti zpracování transakcí a průměrné doby zpracování transakcí velmi dobré (viz test 2A).

TEST 2B

Smyslem tohoto testu je zjistit vliv aplikace modulu vstupního řízení a monitorovacího modulu na průměrnou dobu zpracování transakce u nepřetíženého systému.

Vstupní parametry:

- počet databázových operací v transakci = 5
- počet aperiodických transakcí = 1500
- intervaly, v kterých jsou náhodně posílány transakce do systému: <0,5;25>; <0,5;15>; <0,5;10>; <0,5;7,5>; <0,5;5>, hodnoty v ms



Graf 5.11: Průměrná doba zpracování transakce

Komentář výsledků:

Z grafu č. 5.11 lze vyvodit, že použití algoritmu vstupního řízení nemá výrazný vliv na průměrnou dobu zpracovávané transakce. Průměrná doba zpracování transakce je v mezním intervalu <0,5;25> při použití algoritmu vstupního řízení kratší. Tento jev je způsoben tím, že vykonání algoritmu vstupního řízení je umístěno do části programu, kdy je zakázáno přepnutí kontextu, a proto během velmi krátké doby nejsou do systému generovány další transakce.

6. VÝSLEDKY A VYHODNOCENÍ

Testy popsané v kapitolách 5.5 a 5.6 jsou zaměřeny na dvě oblasti. První oblast zahrnuje zjištění, při jakých parametrech dochází k přetížení systému Testbed. Pro testy 2A a 2B bylo nutné najít takový stav systému, kde je již znatelný vliv degradace na propustnost. Tohoto stavu bylo dosaženo postupným zvyšováním počtu vstupních transakcí, konkrétně změnou intervalu generování transakcí. Experimentálně bylo zjištěno, že systém Testbed dokáže úspěšně zpracovat cca 100 % transakcí, pokud přijímá méně než 130 transakcí za sekundu, při dané konfiguraci a množině vstupních transakcí. Při generování 130–400 transakcí za sekundu již některé transakce (v řádu jednotek) promeškávají kritický termín. Při náhodném generovaní transakcí v intervalu 0,5-5 ms, tzn. cca 400 transakcí za sekundu a více lze sledovat významnou změnu vytížení systému. Při dalším zvyšování počtu vstupních transakcí a zpracování 600 a více transakcí za sekundu je zřejmé, že procento úspěšně zpracovaných transakcí klesá lineárně až k 65 %. Výsledkem testů při daném nastavení systému je nalezení vhodné množiny testovacích transakcí, která způsobí přetížení systému.

Druhý směr testování analyzuje, zda je vhodné použít algoritmus vstupního řízení jako prevenci proti přetížení systému. Rozhodování, kdy a která transakce není do systému vpuštěna, je založeno na informacích ze systému Testbed, proto také mluvíme o algoritmu vstupního řízení založeného na zpětné vazbě systému. Na základě testu 1A, byla definována vstupní množina transakcí. Výsledky testu 2A potvrdily, že algoritmus vstupního řízení dokáže efektivně zabránit přetížení systému, konkrétně při aplikaci navrženého vstupního řízení neklesne procento úspěšnosti zpracovaných transakcí pod 85 % a počet zpožděných transakcí klesne minimálně o 56 % v porovnání se systémem bez vstupního řízení. Obecně lze také konstatovat, že vstupní řízení dokáže eliminovat počet zpožděných transakcí na 16 % a méně. Dále bylo zjištěno, že využití vstupního řízení a monitorovacího modulu nemá výrazný vliv na průměrnou délku zpracování transakce. Z výsledku testu 2B vyplývá, že průměrná doba zpracování transakce není negativně ovlivněna aplikací vstupního řízení. Rozdíly doby zpracování transakcí bez využití vstupního řízení a s využitím vstupního řízení jsou zanedbatelné.

7. CELKOVÉ SHRNUTÍ

V disertační práci byl navržen a popsán algoritmus vstupního řízení, jenž je založen na zpětné vazbě systému a zabraňuje jeho přetížení. Chování systému Testbed se vstupním řízením bylo zkoumáno a sledováno během jednotlivých testů. V první fázi testů bylo zjištěno, při kterých parametrech nastavení systému a vstupních transakcích dochází k přetížení systému. Na základě těchto informací bylo možné vhodně připravit parametry systému tak, aby umožňovaly testování algoritmu vstupního řízení. Z výsledků testů 2A lze vyvodit následující závěry: při aplikaci vstupního řízení jednoznačně klesl počet zpožděných transakcí, a to minimálně o 56 % v porovnání se systémem bez vstupního řízení. Obecně lze také konstatovat, že vstupní řízení dokáže eliminovat počet zpožděných transakcí na 16 % a méně. Dále bylo testem 2B zjištěno, že využití vstupního řízení a monitorovacího modulu má zanedbatelný vliv na průměrnou délku zpracování transakcí. Rozdíly doby zpracování transakce bez použití vstupního řízení a s použitím vstupního řízení jsou minimální.

Během disertační práce byla vyvinuta nová verze systému Testbed, moduly predispatcher, dispatcher a integrovány moduly index manager, souběžné řízení a modul pro před-načítání dat. Samotný vývoj systému Testbed pak umožnil korektně otestovat navržený algoritmus vstupního řízení v prostředí reálného času při využití existujícího databázového systému reálného času. Systém Testbed je nyní připraven na další rozšiřování a testování mechanismů pro oblast vstupního řízení. Byla vyvinuta experimentální platforma, která předpokládá možnost dalšího výzkumu v oblasti databázových systémů reálného času.

V disertační práci byly představeny dva operační systému reálného času RTX a VxWorks s odlišnou filosofií a vlastnostmi pro vývoj komplexního softwarového testovacího nástroje. Přestože během vývoje systému Testbed na platformu RTX autor narazil na skutečnosti, které vývoj systému prodloužily (viz kapitola č. 3.2.5), podařilo se využít tohoto prostředí reálného času pro vývoj nové verze systému Testbed v programovacím jazyce C++ při využití objektového návrhu systému. Lze konstatovat, že právě i díky vlastnostem platformy RTX se podařilo vyvinout robustní, rozšiřitelný systém Testbed, který může sloužit následnému výzkumu databázových systémů reálného času.

Výsledky výzkumu a vývoje systému Testbed, na kterých autor práce participoval, byly průběžně publikovány a jejich seznam lze nalézt ve výčtu tvůrčích a publikačních aktivit. Konkrétně jde o publikace (1), (2), (3), (4), (5), (6), (7), které popisují základní verzi systému Testbed. Výsledky výzkumu vstupního řízení byly prezentovány v publikacích (8), (9), (10) a (11) zabývajících se algoritmy přidělováním priorit a specifiky při analýze a návrhu databázových systému reálného času.

Navržený algoritmus vstupního řízení jednoznačně pomůže předcházet přetížení a degradaci systému Testbed či obecně databázových systémů reálného času a lze tedy konstatovat, že cíl práce byl splněn.

8. DALŠÍ MOŽNÉ ROZŠÍŘENÍ A VYUŽITÍ SYSTÉMU

Oblast vstupního řízení lze dále zkoumat v souvislosti s protokoly pro fixní přidělování priorit, či také v kombinaci s dynamickými protokoly přidělování priorit. Přínosným výzkumem by bylo praktické porovnání existujících mechanismů vstupního řízení pro dynamický protokol přidělování priorit EDF, popřípadě porovnání protokolů AED, AEVD a AAP. Další rozšíření systému může spočívat nejen ve vývoji a testování nových algoritmů vstupního řízení a porovnání s existujícími přístupy, ale také ve zkoumání ostatních součásti systému, tzn. oblasti přidělování priorit, souběžného řízení, indexování dat, buffer managementu a dalších. Využití systému Testbed a případného navazujícího výzkumu lze rozdělit na analytický a aplikační směr.

Analytický směr

Vstupní řízení u systémů ve víceprocesorovém prostředí

- analýza možností využití systému ve víceprocesorovém prostředí
- změna architektury systému, úprava algoritmu vstupního řízení
- testování v reálném prostředí

Testbed a jeho možnosti/omezení použití jako souborového systému pro embedded systémy

- studie o obecné použitelnosti RTDBS jako souborového systému
- skutečnosti, které brání nasazení systému Testbed jako souborového systému a následné provedení úprav systému
- případná implementace jako část operačního systému

Změna databázového schématu (modelu) systému Testbed

- objektový nebo síťový, post-relační[34], případně porovnání s existujícím modelem
- porovnání ukládání dat u daných databázových schémat v praxi

Analýza a dolování dat

- zjišťování skrytých závislostí nástrojem pro dolování dat, například nástrojem SAD vyvinutým na Katedře informatiky Vysoké školy báňské - Technické univerzity v Ostravě

Technologie Firebird - mutliversioning[35] a jeho možnosti použití

- výhody, nevýhody a omezení při využití tohoto konceptu pro systém Testbed
- implementace a porovnání s alternativními přístupy

[34] CACHÉ. Otázky a odpovědi o systému Caché. *www.intersystems.cz* [online]. ©1996-2012 [cit. 2011-12-15]. Dostupné z: http://www.intersystems.cz/cache/cache-qa.htm.

[35] ROKYTSKYY, Roman. A not-so-very technical discussion of Multi Version Concurrency Control. *www.firebirdsql.org* [online]. ©2000-2012 [cit. 2011-12-15]. Dostupné z: http://www.firebirdsql.org/en/multi-version-concurrency-control/.

Aplikační směr

Vytvoření speciálního hardwaru simulujícího klienty

- pro zajištění reálných vstupních transakcí do systému Testbed by bylo vhodné simulovat klienty pomocí speciálního hardware nebo využitím real-time TCP/IP stack

Analýza systému pomocí speciálního softwaru – LDRA či IBM purify

- dynamická analýza kódu, kontrola paměti na tzv. stack overflow[36]

Portace a nasazení systému Testbed na jiný operační systém reálného času (Qnx, RT-Linux nebo MQX)

- využití nástroje profiler, optimalizace kódu, případně refaktorovaní kódu[37] a vyladění systému jako celku[38]
- provedení zátěžových testů, zjištění mezních hodnot

Možnost propojení se softwarovým nástrojem od Martina Hrubého (45) sloužícím k univerzálnímu připojení k databázovému systému

- výhody či nevýhody připojení, decimace dat
- postupy nutné k nasazení a otestování na reálném systému

Použití experimentálního systému Testbed pro aplikaci fotbal robotů

- analýza a případné nasazení systému Testbed, případně jiného systému s vhodným nastavením

Propojení systému Testbed s projektem JumboMem

- distribuované využití paměti
- možnost distribuce daného systému na různých úrovních[39]

[36] FELDMAN, Howard. Modern Memory Management, Part 2. In: *onlamp.com* [online]. November 23, 2005 [cit. 2011-12-15]. Dostupné z: http://onlamp.com/pub/a/onlamp/2005/11/23/memory-management-2.html.

[37] MARTIN FOWLER. Refactoring Home Page. *Martinfowler.com* [online]. [cit. 2011-12-15]. Dostupné z http://martinfowler.com/refactoring/.

[38] DAVID WALKER. Optimizing Compilers. *cs.princeton.edu* [online]. [cit. 2011-12-15]. Dostupné z: http://www.cs.princeton.edu/courses/archive/spr03/cs320/notes/loops.pdf.

[39] SCOTT PAKIN. JumboMem [software]. [přístup 15. prosince 2011]. Dostupné z: http://www.ccs3.lanl.gov/~pakin/software/jumbomem/.

Seznam Obrázků

Seznam Tabulek

Seznam Grafů

Seznam použité literatury

1) ABBOTT, R. - GARCIA-MOLINA, H. Scheduling Real-time Transactions: a Performance Evaluation. In: *Proceeding of 14th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1992, s. 1-12. ISBN 0-934613-75-3.

2) CINGISER, Dipippo Lisa et al. Scheduling and Priority Mapping for Static Real-Time Middleware. In: *Journal Real-Time Systems - Special issue on challenges in design and implementation of middlewares for real time*. Norwell, MA, USA: Kluwer Academic Publishers, 2001, s. 152-182. ISBN 0922-6443.

3) MAYMIR-DUCHARME, F. Dynamic priorities, priority scheduling and priority inheritance. In: *Proceeding IRTAW '90 Proceedings of the fourth international workshop on Real-time Ada issues*. New York, NY, USA: ACM, 1990, s. 39-45. ISBN 0-89791-375-2.

4) LU, Chenyang, et al. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms. In: *The International Journal of Time-Critical Computing Systems*. Norwell, MA, USA: Kluwer Academic Publishers, 2002, s. 85-126. ISSN 0922-6443.

5) LEHMAN, J. - Tobin, J. - CAREY, Michael. A Study of Index Structures for Main Memory Database Management Systems. In: *Proceeding VLDB '86 Proceedings of the 12th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1986, s. 294-303. ISBN 0-934613-18-4.

6) VIGUIER, R. Igor - DATTA, Anindya. Buffer Management in Active, Real-Time Database Systems - Concepts and an Algorithm. In: *Proceeding ARTDB '97 Proceedings of the Second International Workshop on Active, Real-Time, and Temporal Database Systems*. London, UK: Springer-Verlag, 1997, s. 141-185. ISBN 3-540-65649-9.

7) KAM-YIU, Lam. - KUO, Tei-Wei - S.H. LEE, Tony. Strategies for resolving inter-class data conflicts in mixed real-time database systems. In: *Journal of Systems and Software*. New York, NY, USA: Elsevier Science Inc., 2002, s. 1-14. ISBN 0164-1212.

8) ULUSOY, Özgür, BUCHMANN, Alejandro A real-time concurrency control protocol for main-memory database systems. In: *Journal Information Systems, Volume 23*. Oxford, UK: Elsevier Science Ltd., 1998, s. 109-125. ISBN 0306-4379.

9) DATTA, A. H. - SON, S. H. A Study of Concurrency Control in Real-Time, Active Database Systems. In: *Journal IEEE Transactions on Knowledge and Data Engineering, Volume 14*. Piscataway, NJ, USA: IEEE Educational Activities Department, 2002, s. 465-484. ISSN 1041-4347.

10) BESTAVROS, A. - NAGY, S. Value-cognizant admission control for RTDB systems. In: *Proceedings of the 17th IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 1996, s. 230. ISBN 0-8186-7689-2.

11) Lortz, Victor B. - Kang, G. Shin - Kim, Jinho. MDARTS: A Multiprocessor Database Architecture for Hard Real-Time Systems. In: *Journal IEEE Transactions on Knowledge and Data Engineering, Volume 12*. Piscataway, NJ, USA : IEEE Educational Activities Department, 2000, s. 621-644. ISSN 1041-4347.

12) RAMAMRITHAM, K. - DIPIPPO, L. Real-Time Databases and Data Services. In: *Real-Time Systems Journal, vol. 28*. Norwell, MA, USA: Kluwer Academic Publishers, 2004, s. 179-215. ISSN 0922-6443.

13) ALDARMI, S. A.. *Real-Time Database Systems: Concepts and Design*. United Kingdom: The University of York, Department of Computer Science, 1998.

14) NAGY, S. - BESTAVROS, A. Admission control for soft-deadline transactions in ACCORD. In: *Real-Time Technology and Applications Symposium*. Boston, MA, USA: Boston University, 1997, s. 160-165. ISBN 0-8186-8016-4.

15) ADELBERG, B. - GARCIA-MOLINA, H. - KAO, B. Emulating Soft Real-Time Scheduling Using Traditional Operating System Schedulers. In: *Real-Time Systems Symposium*. Stanford, CA, USA: Stanford University, 1994, s. 292-298. ISBN 0-8186-6600-5.

16) YEUNG, Chim-fu. - HUNG, Sheung-lun. A new deadlock detection algorithms for distributed real-time database systems. In: *Proceedings of the 14TH Symposium on Reliable Distributed Systems*. Washington, DC, USA: IEEE Computer Society, 1995, s. 146-153. ISBN 0-8186-7153-X.

17) SIVASANKARAN, R. - PURIMETLA, B.. *Design of RADEx - Real-time active Database Experimental System*. United States: Department of Computer Science, University of Massachusetts, Amherst, 2005.

18) KRISHNA, C.M. *Real-Time Systems*. US: Mcgraw-Hill College, 1996. ISBN 0-07-057043-4.

19) REINHARD, W., et al. The Worst-Case Execution Time Problem— Overview of Methods and Survey of Tools. In: *ACM Transactions on Embedded Computing Systems, Volume 7*. New York, NY, USA : ACM, 2008, s. 36. ISSN 1539-9087.

20) KROL, V. *Metody ověřování vlastností real-time databázového systému s použitím jeho experimentálního modelu*. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2006. PhD. Thesis.

21) ŠARMANOVÁ, J. *Teorie zpracování dat*. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 1997. ISBN 80-7078-491-1.

22) STANKOVIC, J. - ZHAO, W. On Real-Time Transactions. In: *ACM SIGMOD Record - Special Issue on Real-Time Database*. New York, NY, USA: SIGMOD RECORD, 1988, s. 4-18. ISSN 0163-5808.

23) GRAY, J. - REUTER, A. *Transaction Processing : Concepts and Techniques*. San Francisco: Morgan Kaufmann Publishers, Inc., 1993. ISBN 1-55860-190-2.

24) ČERNOHORSKÝ, J. - SROVNAL, V. Systémy reálného času. In: *AT&P journal, Issue 6-8, Volume 12*. Bratislava: ATP Journal, 2010, s. 20-21. ISSN 1335-2237.

25) DENNING, P. J. et. al. Optimal Multiprogramming. In: *Proceedings of Acta Informatica 7, Volume 2*. Trier, Deutschland: Springer-Verlag, 1976, s. 197-216. .

26) TAY, C. Y., et. al. Locking performance in centralized databases. In: *ACM Transactions on Database Systems*. New York, NY, USA: ACM, 1985, s. 415-462. ISSN 0362-5915.

27) Dan, A. - Towsley, D.F. - Kohler, W.H. Modelling the Effect of Data and Resource Contention on the Performance of Optimistic Concurrency Control Protocols. In: *Proceeding 4rh Conference on Data Engineering*. Los Angeles, CA , USA: IEEE Computer Press, 1988, s. 418-425. ISBN 0-8186-0827-7.

28) KOLÁŘ, Petr. *Operační systémy* [online]. 2005 [cit. 2011-12-15]. Dostupné z: http://www.kai.vslib.cz/~kolar/os/

29) LIU, C. L. - LAYLAND, W. J. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. In: *Journal of the ACM (JACM), Volume 20 Issue 1*. New York, NY, USA: ACM, 1973, s. 46-61. ISSN 0004-5411.

30) Haritsa, J.R. - Livny, M. - Carey, M.J. Earliest Deadline Scheduling for Real-Time Database Systems. In: *In Proceedings of IEEE Real-Time Systems Symposium '1991*. San Antonio, TX, USA:

IEEE Computer Society Press, 1991, s. 232-243. ISBN 0-8186-2450-7.

31) KAM-YIU, Lam et al. Designing Inter-Class Concurrency Control Strategies for Real-time Database Systems with Mixed Transactions. In: *Proceedings of the 12th Euromicro conference on Real-time*. Washington, DC, USA: IEEE Computer Society Press, 2000, s. 47-54. ISBN 0-7695-0734-4.

32) Ramamritham K. - DiPippo L. Real-Time Databases and Data Services. In: *Real-Time Systems Journal, Vol. 28, Issue 2-3*. Norwell, MA, USA: Kluwer Academic Publishers, 2004, s. 179-215. ISSN 0922-6443.

33) CHLUBNA, Ondřej. *Analýza a implementace nových algoritmů souběžného řízení RT databází*. Ostrava, 2007. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky.

34) STRATIL, Vít. *Analýza a implementace indexovacích mechanismů v RT databázích*. Ostrava, 2007. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky.

35) KUBÍČEK, Jan. *Analýza a implementace ukládání dat RT databází na externí média*. Ostrava, 2007. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky.

36) L'ECUYER, Pierre. UNIFORM RANDOM NUMBER GENERATORS: A REVIEW. In: *Proceedings of the 1997 Winter Simulation Conference*. Atlanta, GA, USA: ACM, 1997, s. 127-134 . ISBN 0-7803-4278-X .

37) Martin Kot Modeling real-time database concurrency control protocol two-phase-locking in Uppaal. In: *Proceedings of the InternationalMulticonference on ComputerScience and Information Technology*. London, UK: Springer London, 2009, s. 673-678. ISBN 978-83-60810-14-9.

38) Eckel Bruce *Myslíme v jazyku C++*. : Grada Publishing, 2000. ISBN 80-247-9009-2.

39) McCabe, Thomas A Complexity Measure. In: *IEEE transactions on software engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976, s. 308-320. ISSN 0098-5589.

40) Kuo-Chung, Tai. A program complexity metric based on data flow information in control graphs. In: *ICSE '84 Proceedings of the 7th international conference on Software engineering*. Orlando, Florida, USA : IEEE Press, 1984, s. 11. ISBN 0-8186-0528-6 .

41) DataPartner *Ardence RTX Doplňek reálného času pro řízení pomocí Windows* [online]. 2011 [cit. 2011-12-15]. Dostupné z: http://www.datapartner.cz/sites/default/files/RTX_CZ.pdf

42) Microsoft, Adrence *Hard Real-Time with Ardence RTX on Microsoft Windows XP and Windows XP Embedded* [online]. 2002 [cit. 2011-12-15]. Dostupné z: http://msdn.microsoft.com/en-us/library/ms838340%28v=winembedded.5%29.aspx

43) MORYC, P. *Měření procesů reálného času v operačním systému RTLinux*. Ostrava: Vysoká škola báňská - Technická univerzita Ostrava. Fakulta elektrotechniky a informatiky, 2007. PhD. Thesis.

44) MORYC, P. - CERNOHORSKY, J. Task jitter measurement under RTLinux and RTX operating systems, comparison of RTLinux and RTX operating environments. In: *Proceedings of the International Multiconference on Computer Science and Information Technology*. Wisła, Poland: IEEE, 2008, s. 703-709. ISBN 83-922646-0-6.

45) HRUBÝ, Martin. *Design softwarového nástroje pro ukládání měřených dat*. Ostrava, 2007. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky.

VÝČET TVŮRČÍCH A PUBLIKAČNÍCH AKTIVIT

PUBLIKAČNÍ ČINNOST V RÁMCI PŘÍPRAVY K ŘEŠENÍ DISERTAČNÍ PRÁCE

1) KROL V., CERNOHORSKY J., POKORNY J. Towards the Evaluation of Algorithms used in Real-Time Databases. In: *Proceedings of the 7th WSEAS International Conference: Automatic Control, Modeling and Simulation*. Prague, 2005, s. 67-71, ISBN 960-8457-12-2.

2) POKORNÝ J., KRÓL V., ČERNOHORSKÝ J. Návrh databázového systému v prostředí operačního systému založeného na specifikaci real-time. In: *Sborník přednášek Tvorba Software 2005*. Ostrava, 2005, str. 207-210. ISBN 80-86840-14-X.

3) KROL V., POKORNY J., CERNOHORSKY J. Architecture of experimental real-time databases for embedded systems. In: *Proceedings of IFAC Workshop on Programmable Devices and Embedded Systems PDeS 2006*. Brno, 2006, s. 384-388. ISBN 80-214-3130-X.

4) POKORNÝ J., KRÓL V. Testování rt algoritmů při zpracování transakcí v experimentálním real-time databázovém systému. In: *Proceedings of the 7th International Scientific - Technical Conference - PROCESS CONTROL 2006*. Kouty nad Desnou, 2006, s. 187-192. ISBN 80-7194-860-8.

5) KROL V., POKORNY J. The V4DB project - support platform for testing the algorithms used in real-time databases. In: *Proceedings of the 10th WSEAS Int. Conf. on Computers*. Athens, Greece, 2006, s. 185-191, ISBN 960-8457-47-5.

6) KROL V., POKORNY J. Design of experimental platform for testing real-time database transaction processing. In: *3rd IFAC Workshop on Discrete-Event System Design* (DesDes06). Zielona Gora:University of Zielona Gora, Poland, 2006. ISBN 83-7481-035-1.

7) KROL V., POKORNY J. Design of V4DB - Experimental Real-Time Database System. In: *The 32nd Annual Conference of the IEEE Industrial Electronics Society*. Paris, France, 2006, s. 126-131. ISBN 1-4244-0390-1.

8) POKORNY J., KROL V. The V4DB Testbed – Evaluating of Real-Time Database Transaction Processing Strategies. In: *Proceedings of the 11WSEAS International Conference on Computers*. Agios Nikolaos, Crete Island, Greece, 2007, s. 20-25. ISBN 978-960-8457-92-8.

9) POKORNY J. The V4DB Project Testing: Priority Assignment Strategies. In: *International Workshop Control and Information Technology*. Ostrava, 2007, s. 57-60. ISBN 978-80-248-1567-1.

10) POKORNY J. The V4DB Real-time Database System: Admission Control and Priority Assignment. In: *Proceedings of the 5th annual workshop, WOFEX 2007*. Ostrava, 2007, s.212-214. ISBN 978-80-248-1571-8.

11) POKORNY J., FRANEK, Z. Databases in Real-Time: Experimental Research. In: *8th International Conference on Information Technology and Telecommunication*. Galway:Galway-Mayo Institute of Technology, Ireland, 2008, s.225-228. ISSN 1649-1246.

PUBLIKAČNÍ ČINNOST OSTATNÍ

1) POKORNY J., KROL V. The specific way of data storage implementation in the robot soccer game. In: *IFAC Workshop on Programmable Devices and Embedded Systems PDeS 2006.* Brno, 2006, s. 460-463. ISBN 80-214-3130-X.
2) KOZANY, J., TUCNIK, P., POKORNY, J., LUKAS, D., SROVNAL, V. Software structure of a control system for the mirosot soccer game. In: *Ed. Norman Weiss, Norbert Jesse, Bernd Reusch*. Dortmund: University of Dortmund, 2006, s. 183-188. ISBN 3-00-019061-9.

ŘEŠENÉ PROJEKTY

GAČR, GA102/06/1742, 2006-2008.

Černohorský J., Król V., „Experimentální testovací systém pro real-time databáze“. VŠB-TU Ostrava, SU OPF Karviná.

GAČR, GA102/05/H525, 2006-2007.

Pokorný M., „Racionalizace studia doktorského studijního programu na FEI VŠB-TU Ostrava“. VŠB-TU Ostrava.

GAAV ČR, 1ET101940418

Horák, B, „Strategické řízení systémů s multiagenty“, VŠB-TU Ostrava.

CAK (Centrum Aplikované Kybernetiky), 1M0567, druhá polovina roku 2005

Projekt MŠMT na podporu výzkumu a vývoje

SEZNAM PŘÍLOH

Příloha A: Popis struktury TRX_INFO

Příloha B: CD obsahující následující informace:

- zdrojové kódy vyvinutého experimentálního databázového systému reálného času Testbed
- prezentované grafy včetně zdrojových dat ve formátu *.ods a *.csv

PŘÍLOHA A

Popis struktury TRX_INFO

Název atributu	Popis atributu
orig_task_id	čitelný identifikátor procesu
gen_id	čitelný identifikátor generátoru, který danou transakci posílá do systému
task_id	systémový identifikátor procesu zpracovávající danou transakci
trx_id	nastaveno při přidání transakce do struktury TRX_INFO, pořadové číslo transakce
trx_id_lock	identifikátor první transakce, se kterou je daná transakce v konfliktu
timeTick_PreDispatch	čas přijetí transakce modulem predispatcher
timeTick_LoadBalance_Start	čas před vstupním řízením, okamžik před spuštěním funkce LoadBalance()
timeTick_LoadBalance_End	čas po zpracování funkce LoadBalance()
timeTick_Dispatch_first	čas prvního dispatchování transakce
timeTick_Dispatch_last	čas posledního dispatchování transakce, v případě spuštění pouze jednou se rovna času prvního dispatch transakce
timeTick_TrxProcess_Start_first	čas prvního zpracovávání transakce modulem Transaction managerem
timeTick_TrxProcess_Start_last	čas posledního spuštění úlohy zpracovávající transakci v modulu Transaction Manager, pokud byla spuštěna opakovaně; v případě pouze jednoho spuštění se rovná okamžiku prvního spuštění
timeTick_TrxProcess_Parser	čas před parsováním na jednotlivé transakce
timeTick_TrxProcess_CommandExecutor	čas před dělením databázových operací na R a W
timeTick_TrxProcess_CC_start	čas před obalením zámky
timeTick_TrxProcess_CC_End	čas po obalení zámky
timeTick_TrxProcess_BufferMan_Start	čas před zpracováním modulem Buffer Managerem
timeTick_TrxProcess_BufferMan_End	čas po práci s Buffer Managerem
timeTick_TrxProcess_End	čas ukončení trx_task, tzn. po ukončení poslední operace, před provedením taskUnlock()
timeTick_TrxProcess_Commit	čas committu transakce, nastaveno hned po nastavení příznaku committed

time_Execution_ticks	jestliže byla transakce commitnuta, je hodnota nastavena na rozdíl doby, kdy byla commitnuta a časem, kdy začala být zpracováváná predispatcherem, pokud není commitnuta, je hodnota nastavena na 0
deadline_satisfied	pokud byla transakce commitnuta a úspěšně ukončena v termínu (hodnoty 1 nebo 0)
committed	jestli byla transakce commitnuta, tzn. jestli se proces dostal do stavu commit
aborted	jestli byl poslední známý stav transakce aborted
restarted	jestli byl poslední známý stav transakce restarted
criticality	vlastnost transakce z trx definice
period	vlastnost transakce z trx definice
priority_start	vlastnost transakce, nastavena během dispatche transakce, v případě pouze jednoho spuštění se priority_start rovna priority_end
priority_end	hodnota pripority transakce, nastavena během dispatche transakce, v případě pouze jednoho spuštění se priority_end rovna priority_start
restart_count	počet restartů dané transakce
blocked_count	v případě výskytu konfliktu zvýšeno o 1, konflikt však mohl byt detekován rychle vícekrát pro jednu transakci